http://dx.doi.org/10.3745/KIPSTD.2011.18D.6.443

모델기반 컴포넌트 정제 과정의 행위 일관성 검증을 위한 변환기

장 훈[†] · 박 민 규[††] · 최 윤 자[†††]

요 약

모델 중심 컴포넌트 기반 개발 방법은 개발 대상 시스템을 하나의 추상컴포넌트로 보고, 재귀적인 컴포넌트의 분화(decomposition)와 정제(refinement) 과정을 통하여 물리적인 컴포넌트를 도출해내는 하향식 개발방식이다. 본 연구에서는 모델 기반 컴포넌트 개발기법을 근간으로 한 컴포넌트 정제 과정에서 정제 전·후의 추상컴포넌트들 간의 행위 일관성을 정형분석하기 위한 모델 변환기를 개발하였다. 이 모델변환기는 컴포넌트의 분화와 정제 전 과정에 걸쳐 사용되어 정제 전·후 컴포넌트 간의 상호작용 오류로 인한 잠재적인 결함을 조기에 발견하고 해결하는데 기여할 수 있다. 본 논문은 추상컴포넌트의 각 구성요소들을 정형검증기 SPIN의 구성요소로 변환하기 위해 사용된 변환방법들과 변환기의 구성요소들을 소개한다. 개발된 변환기는 자동차 거울조종시스템, 무선센서네트워크를 위한 운영체제 등의 사례연구에 적용되어 그 효용성을 입증하였다.

키워드 : 일관성 검증, 추상 컴포넌트, 모델 검증법

A Model Translator for Checking Behavioral Consistency of Abstract Components

Hoon Jang[†] · Mingyu Park[††] · Yunja Choi[†††]

ABSTRACT

Model-based Component development methodologies consider the whole system as an abstract component and develop physical components through recursive decomposition and refinements of abstract components in a top-down manner. We developed a model translator that can be used to formally verify interaction consistency among components, especially the interaction behavior between before- and after- refinements of abstract components. This translator can be used to identify potential problems in the refinement process so that problems can be addressed from the early stage of development. This paper introduces our translation approach and the organization of the translator. The translator has been applied to two case studies to show its usefulness.

Keywords : Interaction Consistency, Abstract Component, Model Checking

1. 서 론

모델중심 컴포넌트 기반 개발 방법(Model-driven component-based development)은 시스템을 분화, 정제하여 물리 컴포넌트들을 도출해 내는 하향식 개발 방식이다. 이러한 방식은 시스템 개발 초기부터 모델검증기법을 도입하여 품질을 보증하고, 모델과 구현코드와의 연속성을 유지할 수 있다는 장점을 가진다. 그러나, 이러한 장점은 세분화 과정의 정확성과 일관성의 유지를 전제로 하며, 모델검증 도구의 보조 없이는 실효성을 보장할 수 없다.

예를 들어, 정제 전 컴포넌트 A가 정제 후, 컴포넌트 B에 정의된 서비스를 사용하는 방식으로 추상컴포넌트간에 분화(decomposition)와 정제(refinements)가 일어났을 때, A가 B의 서비스를 사용하는 방식과 B가 서비스를 제공하는 방식 사이에 불일치가 발생하면 프로세스(process) 교착상태와 같은 심각한 문제를 야기시킬 수 있다. 특히, 수십 개의 컴포넌트들이 상호작용하여 서비스를 구현하는 경우, 행위양식의 복잡도는 크게 증가하게 되며, 체계적인 검증 방법 및 자동화 도구의 지원 없이는 엄밀한 품질관리와 검증이 어렵다.

※ 이 논문은 2009년~2010년 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(2010-0017156).
† 정 회 원 : LG전자 HA 제어연구소 연구원
†† 준 회 원 : 경북대학교 전자전기컴퓨터학부 공학석사
††† 정 회 원 : 경북대학교 IT대학 컴퓨터학부 조교수
논문접수 : 2011년 4월 25일
수 정 일 : 1차 2011년 6월 11일, 2차 2011년 6월 22일
심사완료 : 2011년 6월 22일

본 연구에서는 모델중심 컴포넌트 기반 개발방법 중 하나인 MARMOT[5]에 기초를 두고 정제 전·후 추상컴포넌트들 간 상호작용의 일관성을 체계적으로 검증하는 방법과 검증 지원 도구를 개발하였다. 행위일관성 검증은 (1) 환경모델의 정의와 자동추출, (2) 정제 후 추상컴포넌트의 행위모델 추출, 그리고 (3) 일관성모델의 정의와 정형모델로의 모델변환기 개발 등으로 지원되었다.

본 연구에서 개발한 모델변환기는 컴포넌트의 정제 전·후의 추상컴포넌트들 간 상호작용의 일관성을 검증하는 것을 목표로 한다는 점에서 기존의 UML 다이어그램 간의 일관성 검증과 차별화된다. 또한, 기존의 방식이 함수 호출 순서와 관련된 일관성에 국한되어 있었던 것에 비하여, 함수 호출시의 매개변수와 함수호출의 결과로 나타나는 액션명세(Action Specification) 등을 모두 다루었다는 점에서 향상되었다. 행위명세의 정형적 정의와 변환을 위하여 액션구문(Action syntax)을 정의하였으며, 변환된 모델은 SPIN 검증기를 이용하여 자동화된 일관성 검증을 실시하게 된다. 개발된 변환기는 자동차 거울조종시스템, 무선센서네트워크 운영체제 등에 적용되어 그 효용성을 입증하였다.

본 논문은 다음과 같이 구성되었다. 1장 서론에 이어 2장에서는 관련연구와 본 연구의 차별성을 논의하고, 3장에서 본 연구의 기반이 되는 모델 중심 컴포넌트 기반 개발방법론 MARMOT의 간략한 소개 및 정제과정에서 컴포넌트간 행위일관성에 대해 정의한다. 4장에서 행위일관성 검증을 위한 모델 변환기를 제안하고, 5장에서는 변환기의 구현 및 적용결과를 분석하며 6장에서 결론을 맺는다.

2. 관련 연구

참고문헌[1]은 모델의 객체지향 분석을 위하여 객체모델, 동적모델, 기능모델 별로 모델을 정형화하고 일관성 점검 규칙을 정의하였으며, Prolog를 사용하여 일관성 규칙을 점검하였다. 참고문헌[2]는 객체모델의 메서드(method)와 상태전이도(state transition diagram) 간의 상호참조를 통한 일관성 검증기를 설계하였다. 또한, 참고문헌[3]은 UML 상태기계 다이어그램을 이용하여 컴포넌트 인터페이스의 행위 호환성 검증을 위한 도구를 개발하였다. 이러한 도구들은 같은 추상수준에서 상호작용하는 두 행위모델 간의 행위일관성을 주로 다루고 있으며, 함수호출관계에만 역점을 두고 있어 구체적인 매개변수의 처리나 호출의 결과에 의한 행위구문 등은 다루고 있지 않다.

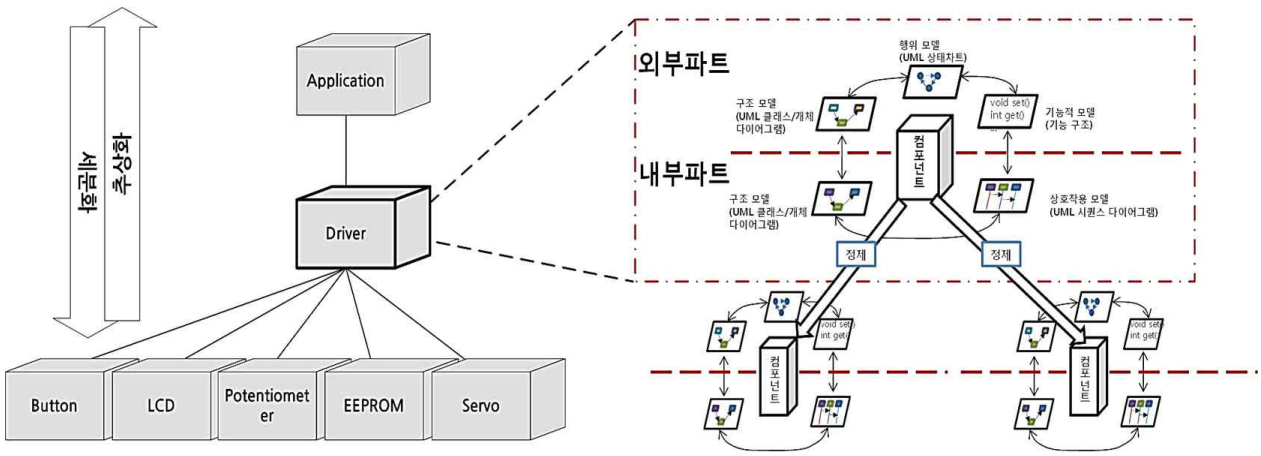
참고문헌[7]에서는 Java언어를 기반으로 한 프로그램(program) 단위의 컴포넌트 모델을 사용하여 시스템을 구축할 수 있는 프레임워크(framework)와 도구를 제공하였다. 참고문헌[8]은 추상 컴포넌트 모델을 정형적으로 정의하고, 단위 컴포넌트인 atomic컴포넌트의 행위와 컴포넌트간의 상호작용, 그리고 상호작용 사이의 우선순위를 적용하여 컴포넌트를 조합, 전체 시스템을 구성하는 방법에 대해 소개하였다. 이러한 방법들은 물리적인 컴포넌트를 대상으로 한 상향식 개발방식에 적합하며, 개발 초기 모델에서부터 구현단계까지 연속적인 개발과정에서의 일관성 검증에 적용할 수 없다.

본 논문에서 소개하는 모델 변환기는 컴포넌트 정제 전·후의 수직적 관계에서의 행위일관성을 검증 대상으로 하며, 초기의 추상컴포넌트 모델에서 최하위의 물리적 컴포넌트의 구현까지 전 과정에 적용하여 컴포넌트의 정제 및 분화의 연속성을 보장하는 데에 목표를 두고 있다. 또한, 행위일관성 검증에 있어서 함수호출관계, 매개변수의 사용, 호출 결과에 의한 행위명세 등을 모두 고려하여 처리하고 있다는 점에서 기존의 방식과 구별된다.

3. 추상 컴포넌트와 정제 프로세스

3.1 모델기반, 컴포넌트 중심 개발 방법론 MARMOT

모델기반, 컴포넌트 중심 개발 방법론인 MARMOT[4],[5]은 내장형 소프트웨어의 개발에 중점을 두어, 추상 컴포넌트의 정제과정을 통한 하향식 컴포넌트 개발방법을 체계화하였다. (그림 1)은 MARMOT의 추상 컴포넌트 모델을 도



(그림 1) 모델기반 컴포넌트 개발

식화한 것이다. 이 개발방식은 개발대상 시스템을 하나의 추상 컴포넌트로 간주하고 세분화와 정제를 이용한 점진적인 컴포넌트의 디자인과정을 통해 단위 물리 컴포넌트를 개발한다.

(그림 1)의 예에서, 자동차 양 옆 거울의 위치를 자동으로 조정하는 소프트웨어를 하나의 추상 컴포넌트인 Application으로 표현하였다. 이 추상컴포넌트는 Driver 추상 컴포넌트로 구체화되어 Application이 제공하는 서비스를 구현하게 되며, 이 Driver는 다시 EEPROM, LCD, Potentiometer, Servo, Button등 각각의 장비와의 인터페이스를 담당하는 드라이버들로 세분화 된다. 그림에서 나타난 최하위 컴포넌트들은 실제 프로그램코드로 구현되는 물리적인 컴포넌트들이며, Application, Driver와 같이 세분화 과정에서 도출되는 컴포넌트들은 물리적인 컴포넌트와 차별하여 추상 컴포넌트로 불린다. 추상컴포넌트의 역할은 추상수준에서의 컴포넌트 행위양식을 정의하고 구체화된 하위컴포넌트들 간의 상호작용을 정의하여 구현된 물리컴포넌트들의 순차적인 조합을 지원하는 것이다.

(그림 1)의 오른쪽 부분은 하나의 추상 컴포넌트를 구성하는 외부모델과 내부모델을 도식화한 것이다. 추상 컴포넌트의 외부모델은 외부에 제공되는 기능들과 행위양식들을 정의하며, 내부모델은 외부에 제공되는 기능들이 구현되는 방식을 컴포넌트의 내부구조와 내부행위양식으로 구분하여 정의한다. 그림의 예제에서 Driver 추상 컴포넌트의 내부모델은 하위 세부 컴포넌트 EEPROM, LCD, Potentiometer, Servo, Button이 어떻게 연계되어 Driver기능을 구현하는지 명세한다. 컴포넌트 구조모델은 UML 클래스다이어그램으로 표현하고, 외부행위모델의 명세는 상태다이어그램을, 내부행위모델은 순차다이어그램을 사용하여 명세한다.

이러한 하향식 개발방식을 통해 추상화와 세분화 수준에 따른 컴포넌트 행위모델을 관리할 수 있고, 각 세분화 과정에서 독립적인 검증이 이루어질 수 있다.

3.2 컴포넌트 행위모델과 일관성모델의 정의

컴포넌트 행위는 Labeled Transition System(LTS)로 표현할 수 있는 상태전이와 상태전이를 유발하는 이벤트, 상태전이의 결과로 수행되는 액션(action)으로 정의한다.

정의 1. A Labeled Transition System (LTS)

LTS는 (S, L, R, I, T)의 쌍으로 구성된다. S는 컴포넌트 상태의 집합, L은 시스템의 행위를 나타내는 레이블 (Lable)의 집합. R은 컴포넌트의 전이관계로서 $R \subseteq S \times L \times S$ 이다. I는 컴포넌트의 초기 상태 집합이고 T는 컴포넌트의 종료 상태 집합이다.

컴포넌트 행위를 표현하는 하나의 프로세스 $P = (S, L, R, I, T)$의 상태 s_i에서 상태 s_{i+1}로 전이는 다음과 같이 정의된다. P가 상태 s_i에 있을 때 L의 원소 l_i가 $(s_i, l_i, s_{i+1}) \in R$를 만족하면 상태전이가 발생하며, 이를 $P \xrightarrow{l_i} P$ 로 나타낸다. 또한, 프로세스의 상태전이는 액션을 생성할 수 있는데, 이를 $(s_i, l_i/l'_i, s_{i+1})$ 혹은 $s_i \xrightarrow{l_i/l'_i} s_{i+1}$로 표현한다. 이는 프로세스가 $l_i \in L$에 의해 s_i에서 s_{i+1}로 전이될 때 액션 $l'_i \in L$를 수행한다는 의미이다. 여기서, 레이블 집합 L은 트리거(trigger) 레이블의 집합 L_t와 액션 레이블의 집합 L_a의 합집합이다. L_t는 컴포넌트가 외부로 제공하는 이벤트들과 해당 전이 발생 조건에 관한 집합이고 L_a는 L_t집합의 한 원소에 의해 전이가 발생했을 때 수행되는 결과 값 전송, 시그널 및 함수 호출에 대응되는 이벤트 전송, 결과값 대입, 사칙연산 등의 액션을 포함한다.

컴포넌트 행위 모델은 역시 LTS로 정의된 환경모델과 조합되어 하나의 닫힌 시스템을 구성한다. 환경모델과 컴포넌트 모델의 조합규칙을 정의 1의 LTS를 이용하여 정형적으로 표현하면 정의 2와 같다.

정의 2. 환경모델과 컴포넌트 모델의 조합규칙

프로세스 $P = (S^p, L^p, R^p, I^p, T^p)$ 와 P의 환경모델 $E = (S^e, L^e, R^e, I^e, T^e)$ 가 존재할 때, P와 E의 조합 $P \uparrow E = (S^p \times S^e, L^p \times L^e, R^p \times R^e, I^p \times I^e, T^p \times T^e)$ 은 다음의 규칙 a, b를 따른다.

규칙 a. $$\frac{s_i^p \xrightarrow{l_i^p} s_{i+1}^p \;,\; s_i^e \xrightarrow{l_i^e/l_i^{e\prime}} s_{i+1}^e}{(s_i^p,\ s_i^e) \xrightarrow{\{\},l_i^e/l_i^{e\prime}} (s_{i+1}^p,\ s_{i+1}^e)} \quad (l_i^{e\prime} = l_i^p)$$

규칙 b. $$\frac{s_i^p \xrightarrow{l_i^p} s_{i+1}^p \;,\; s_i^e \xrightarrow{l_i^e/l_i^{e\prime}} s_{i+1}^e}{(s_i^p,\ s_i^e) \xrightarrow{\{\},l_i^e/l_i^{e\prime}} (s_i^p,\ s_{i+1}^e)} \quad (l_i^{e\prime} \neq l_i^p)$$

규칙 a는 환경모델 E의 전이에 의해 생성된 이벤트가 프로세스 P의 트리거 이벤트(trigger event)로 나타나면 프로세스 P와 환경모델 E가 전이함을 의미하고, 규칙 b는 환경모델 E의 전이에 의해 생성된 이벤트가 프로세스 P의 트리거 이벤트로 나타나지 않으면 P는 현재상태에 머물러있음을 의미한다.

위의 조합규칙에 따라, 일관성 모델을 정의하면 정의 3와 같다.

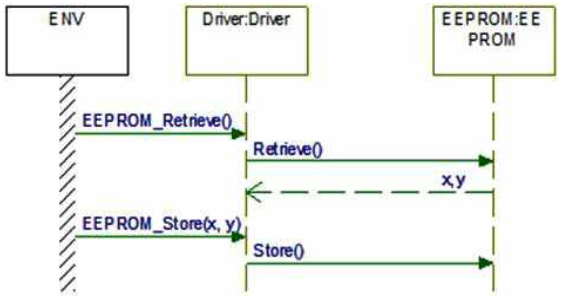
정의 3. 일관성 모델

정제과정에서 일관성 모델은 $P \uparrow E$로 표현되는 닫힌 시스템이다. 여기서, P는 $P = P_1 \mid P_2 \mid \cdots P_n$으로 표현되는 정제 후 컴포넌트들의 외부행위들의 병렬 조합이고, E는 $E_1 \mid E_2 \mid \cdots E_n$으로 표현되는 정제 전 컴포넌트의 내부행위의 병렬 조합이다.

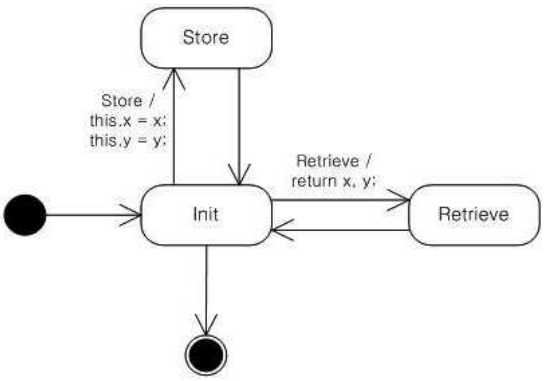
3.3 컴포넌트 모델의 행위 일관성

정제 후 컴포넌트의 외부행위는 정제 전 컴포넌트의 내부

행위 명세에 따라 일관성 있게 동작해야 한다. 예를 들어 (그림 2)는 (그림 1)에서 Driver컴포넌트의 내부행위와 EEPROM컴포넌트의 외부행위를 간략하게 도식화한 것이다.



(a) Driver 컴포넌트의 내부행위



(b) EEPROM 컴포넌트의 외부행위

(그림 2) DRIVER컴포넌트의 구현논리와 EEPROM 컴포넌트의 외부행위

Driver컴포넌트가 환경모델로부터 EEPROM_Retrieve(x,y) 시그널(signal)을 수신 받았을 때, Driver컴포넌트는 이벤트 처리를 위해서 EEPROM에 Retrieve시그널을 보낸 후 반환될 x, y를 기다린다. 하지만 EEPROM 컴포넌트는 Init상태에서 Retrieve시그널을 수신 받기 전, 이미 Termination 상태로 전이되어 종료되었을 수 있다. 이때, Driver컴포넌트는 반환 값을 무한히 대기하게 되며, 이러한 행위 불일치로 인하여 시스템은 EEPROM_Retrieve(x,y) 이벤트를 처리하지 못하게 된다.

따라서 정제 전 컴포넌트의 내부행위가 정제 후 컴포넌트의 외부행위의 환경모델이 될 때, 환경모델에서 무한히 생성되는 이벤트에 대해 컴포넌트의 외부행위양식이 무한히 응답하거나 정상 종료되어야 하고, 이때 컴포넌트의 정제과정에서 정제 전 후 행위일관성은 만족된다고 정의한다.

정의 4. 행위 일관성

컴포넌트 모델은 일관성 모델이 무한히 작동하거나 정상 종료 할 때 행위일관성을 만족한다고 표현한다.

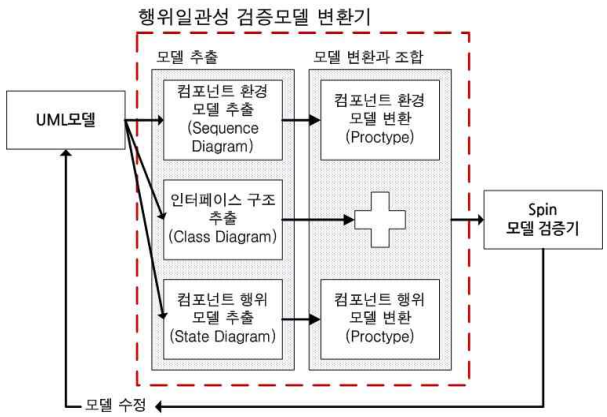
(그림 2)의 예제에서, (a)의 순차다이어그램을 LTS로 변환한 것이 환경모델 E이고, (b)의 상태다이어그램을 LTS로 변환한 것이 일관성모델의 P에 해당한다.

4. 행위일관성 검증을 위한 모델 변환기

4.1 개요

모델 변환기는 UML 다이어그램으로 설계한 컴포넌트 모델을 입력 받고 행위일관성을 검사할 대상 컴포넌트 이름을 선택 받아 정의3에서 정의한 일관성 모델에 해당하는 정형명세를 출력하는 도구이다.

정의3의 컴포넌트 환경 모델 E는 사용자가 선택한 컴포넌트의 내부 행위를 표현하는 순차 다이어그램에서 정제 후 컴포넌트들에 대응되는 각각의 생명선과 상호작용 하는 생명선들을 투영하여 생성한 LTS이다. 변환기는 환경 모델 E와 정제 후 컴포넌트들의 외부행위를 조합하여 일관성 모델을 생성한다.



(그림 3) 행위일관성 검증도구의 구조

(그림 3)은 본 연구에서 개발한 행위일관성 검증도구의 구조를 개략적으로 도식화하였다. 먼저, 정제 전 추상컴포넌트의 내부행위를 표현한 클래스다이어그램, 순차다이어그램으로부터 컴포넌트 환경모델과 인터페이스(interface)를 추출하고, 정제 후 추상컴포넌트의 외부행위를 표현한 상태다이어그램으로부터 대상 컴포넌트의 행위모델을 추출한다. 이러한 환경모델과 대상 컴포넌트 행위모델은 SPIN 모델 검증기의 입력언어인 PROMELA 언어로 변환되어 조합되며, SPIN[6]을 이용한 정형적 모델검증을 수행한다.

4.2 트리(tree) 구조의 컴포넌트 군에서 검증대상 컴포넌트 추출

변환기는 사용자가 정제 프로세스의 개념에 맞게 설계한 UML 다이어그램들의 XML파일(file)을 입력 받아 트리 구조의 컴포넌트 관계를 3장에서 소개한 MARMOT 프레임워크의 컴포넌트간 구체화 관계를 기반으로 추출한다. 이후 추출된 컴포넌트들 중 검증 대상 컴포넌트가 선택되면, 해당 컴포넌트의 하위 컴포넌트들을 추출한다.

4.3 일관성 모델의 정형화

상태다이어그램과 순차다이어그램의 LTS 변환방식은 이미 여러 문헌에서 다루어졌으므로 본 논문에서는 생략한다. 이 절에서는 본 논문에서 제안한 일관성 모델을 정형언어 PROMELA 로 변환하기 위한 방법들을 소개한다.

일관성모델의 정형화는 참고문헌[5]의 MARMOT 컴포넌트 모델의 정형언어 변환규칙을 토대로 이루어졌다. 변환규칙은 MARMOT 컴포넌트 모델을 π-calculus를 이용해 정형적으로 정의하고, 이를 토대로 컴포넌트의 구조와 행위를 PROMELA로 변환한다. 다만, 참고문헌[5]에서의 함수 호출은 파라미터(parameter)를 생략한 단순한 액션만을 지원하지만, 본 변환기에서는 액션 문법을 정의해, 함수 시그네쳐(signature) 전달 뿐만 아니라 함수 인자로서 컴포넌트 애트리뷰트(attribute) 값, 혹은 상수 값을 전달할 수 있게 확장하였다.

4.3.1액션 문법 정의

다음 <표 1>은 액션의 정형적 정의를 위해 정의한 액션 문법의 일부이다. 액션문법을 통해 액션에 파라미터를 생략한 단순한 액션뿐만 아니라 함수 및 시그널 호출, 대입문, 간단한 사칙연산, IF-THEN-ELSE의 조건분기, 결과값 전송을 포함시켰으며, 비정형적으로 정의된 UML상태기계의 액션구문을 보완하고 사용자의 편의성을 높였다.

〈표 1〉 컴포넌트 상태기계의 액션 문법

```
action :            elementary_action
                    | control_action  ;

elementary_         assignment_action ';'
Action :            | return_action ';'
                    | call_action ';'
                    | send_action ';'  ;

control_            IF simple_expression THEN action ENDIF
action :            | IF simple_expression THEN action
                    ELSE action ENDIF  ;

assignment_         attribute_name '=' simple_expression  ;
action :

call_action :       CALL call_expression  ;

call_               classifier_name CLASSIFIER_DELI
expression :        operation_name '(' paramList ')'
                    | operation_name '(' paramList ')'  ;

send_action :       SIGNAL signal_name '(' paramList ')'  ;

return_             RETURN simple_expression
action :            | RETURN paramList

paramList :         /* empty string */
                    | literal
                    | paramList ',' literal
```

다음 <표 2>는 액션구문이 PROMELA 구문으로 전환되는 예제이다.

〈표 2〉 지원되는 액션구문과 변환된 PROMELA구문

	Action Code	PROMELA Code
A	EEPROM! Retrieve();	Driver_EEPROM!pid, RETRIEVE;
B	return x,y;	Driver_Application!pid, RV,x,y;

C	if this.Duration < 30
	then Duration
	= Duration+1;
	else Duration
	= Duration-1;
	endif
		if
		::Duration < 30 → {
		Duration=Duration+1;
		}
		::else → {
		Duration=Duration-1;
		}
		fi;

<표 2>의 예제A는 함수 호출 및 시그널 전송에 대한 예제이다. 예제A의 액션구문은 Driver컴포넌트에서 EEPROM 컴포넌트의 Retrieve()를 호출한다는 의미를 갖고 있다. 이는 PROMELA구문으로 변환될 때, Driver_EEPROM채널(channel)을 통해 Driver컴포넌트의 PID와 호출하고자 하는 이벤트 이름을 EEPROM으로 전송하는 구문으로 변환된다. Driver_EEPROM채널은 EEPROM컴포넌트와 Driver컴포넌트 사이의 메시지 교환을 위한 디폴트 채널이다.

예제B는 결과값 전송에 대한 예제이다. 예제B의 액션구문은 Driver컴포넌트에서 Application컴포넌트로 결과 값 x, y를 전송한다는 의미를 갖고 있다. 이는 PROMELA구문으로 변환될 때, Driver_Application채널을 통해 Driver컴포넌트의 PID와, 이 메시지(message)가 결과값 반환이라는 것을 알리는 RV, 반환하고자 하는 결과값 x, y을 Application포트로 전송하는 구문으로 변환된다.

예제 C는 IF-THEN-ELSE의 조건분기와, 대입문 및 간단한 연산의 변환예제이다.

4.3.2 모델변환

일관성모델의 구성요소인 환경모델과 컴포넌트 외부행위 모델들은 각기 독립적인 PROMELA의 프로세스로 변환되며, 이들간에 디폴트채널을 정의함으로써 독립적인 프로세스간의 상호작용을 지원한다.

```
1   proctype DRIVER( chan DRIVER_EEPROM){
2   ……
3     ::state == 0 → {
4       DRIVER_EEPROM!_pid, RETRIEVE, 0, 0;
5       state=2;
6     }
7     ::DRIVER_EEPROM? [_,eval(RV), _, _]&&state == 2 →
8     {
9       DRIVER_EEPROM? _, _, x, y;
10      state=0;
11    }
12    ::state == 0 → {
13      DRIVER_EEPROM!_pid, STORE,x,y;
14      state=3;
    ……
```

```
45  proctype EEPROM( chan DRIVER_EEPROM){
46  ……
47    ::DRIVER_EEPROM? [_,eval(RETRIEVE), _, _]
48    &&state==0 → {
49      DRIVER_EEPROM?pid, _, _, _ ;
50      DRIVER_EEPROM!pid, RV,x,y;
51      state=0;
52    }
53    ::DRIVER_EEPROM? [_,eval(STORE), _, _]
54    &&state==0 → {
55      DRIVER_EEPROM?pid, _, x, y;
56      this.x= x; this.y= y;
57      DRIVER_EEPROM!pid, ACTIVATE, 0, 0;
58      state=0;
59    }
    ……
```

위 예제는 (그림 1)의 모델에서 일관성 모델을 EEPROM 관점으로 추출, 변환한 결과로, 추출된 일관성 모델의 상세 디자인은 (그림 2)에 나타난 바와 같다. 환경모델 DRIVER와 컴포넌트 외부행위모델 EEPROM은 각각 순차다이어그램과 상태다이어그램의 LTS으로부터 독립적인 PROMELA의 프로세스(proctype)로 변환되며, 두 프로세스 사이에는 디폴트채널 DRIVER_EEPROM을 정의해 메시지를 주고받는다. (그림 2)의 (a)에서 DRIVER가 Retrieve이벤트를 호출하는 행위는 3~6줄과 같이 디폴트채널을 통해 EEPROM에게 RETRIEVE시그널을 전송하는 구문으로 변환된다. (b)에서 외부행위모델 EEPROM이 Retrieve이벤트를 처리하는 행위는 47~52줄과 같이 디폴트채널을 통해 DRIVER에게 RV 메시지를 보내는 구문으로 변환된다.

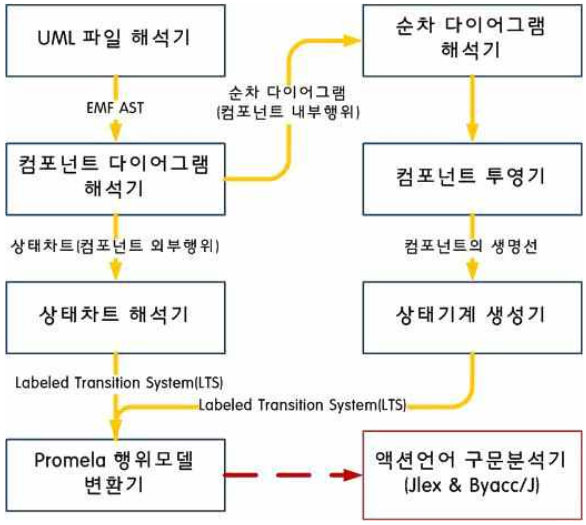
5. 구 현

5.1 개발 환경

행위일관성 검증모델 변환기는 이클립스 모델링 프레임워크(EMF)를 기반으로 한 이클립스 플러그인(plug-in)으로 구현하였다.

5.2 도구 구성

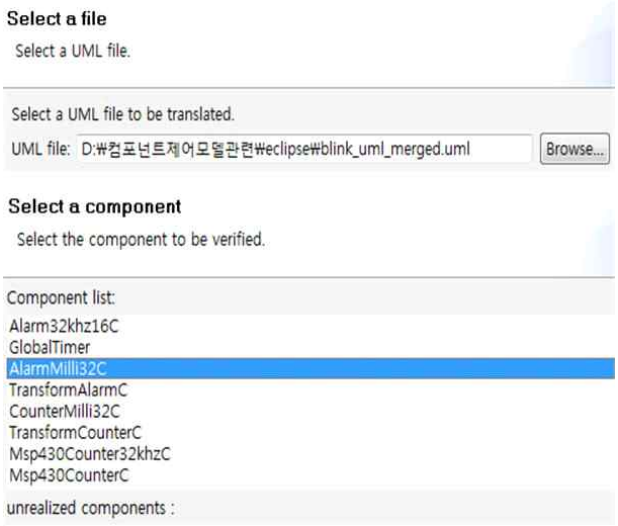
행위일관성 검증모델 변환기의 내부는 (그림 4)과 같이 구성되었다. 모델 변환기는 UML모델을 XML파일의 형태로 사용자 인터페이스를 통해 검증대상 컴포넌트를 입력 받는다. 입력 받은 UML모델은 UML모델 해석기를 거치는데 UML모델 해석기는 입력 받은 XML파일을 이클립스 모델링 프레임워크(EMF)의 추상구문구조(AST)로 변환시킨다. EMF AST는 컴포넌트 다이어그램 해석기에서 컴포넌트의 외부행위에 따른 상태다이어그램과 내부행위에 따른 순차다이어그램으로 분리되고, 이는 다음 단계에 따라 각각 처리된다. 우선 컴포넌트 외부행위로부터 추출된 상태다이어그램은 상태 해석기에서 LTS로 변환된다. 순차 다이어그램은 4절의 순차 다이어그램의 변환방법을 적용하는데, 우선 순차 다이어그램 해석기를 통해 컴포넌트 내부행위로 추출되고, 컴포넌트 투영기에서 투영되어 컴포넌트 각각의 생명선을 중심으로 추출되어 상태기계 생성기에서 LTS로 변환된다. 이렇게 각각의 과정을 통해 변환된 LTS는 PROMELA 행위모델 변환기에서 PROMELA로 변환된다.



(그림 4) 행위일관성 검증도구의 구성요소

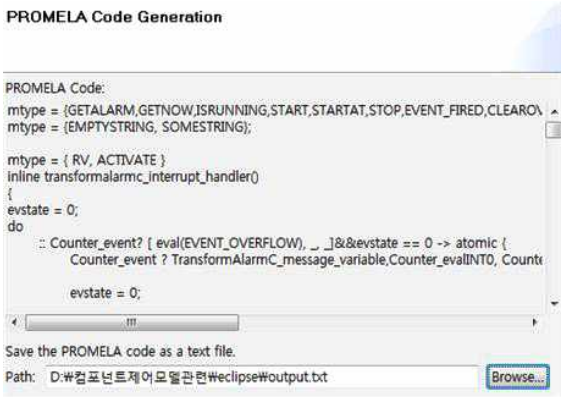
5.3 도구 수행

본 도구는 총 3개의 화면을 제공한다. 첫번째 화면에서 XML 파일을 선택한 후, 두번째 화면에서 행위 일관성을 검증하고자하는 컴포넌트를 선택한다.



컴포넌트를 선택하면, 다음의 화면에서 PROMELA로 변환된 일관성모델의 출력화면을 볼 수 있다. 하단의 Browse 버튼(button)을 눌러서 PROMELA 명세를 저장하고자 하는 경로를 지정하고, Finish버튼을 눌려 파일을 저장한다.

마지막으로 생성된 PROMELA 명세를 대상으로 모델 검증기 SPIN을 이용해 검증을 수행한다.



5.4 모델 검증기 SPIN

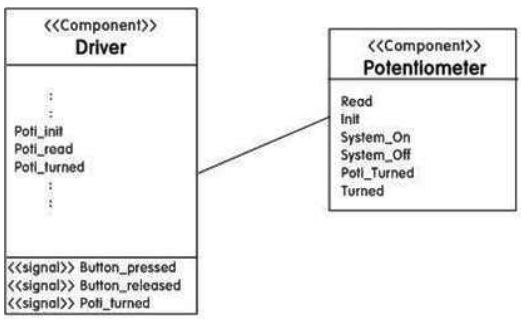
SPIN 모델 검증기[6]는 통신 프로토콜(protocol)과 같이 동시성을 갖는 소프트웨어(software) 모델의 검증에 사용되는 도구이다. 시스템은 비결정적 오토마타(automata)를 통한 비동기적 분산 알고리즘(algorithm)의 설계를 지원하는 PROMELA 모델링(modeling) 언어로 표현되고 검증된다. SPIN은 시스템 상태공간의 탐색을 통하여 잠재적 오류를 찾아내고, 그 오류가 어떤 경로로 발생할 수 있는지에 대한 반례를 생성하여 오류의 원인을 파악할 수 있도록 지원한다. 또한, 모델 시뮬레이터(simulator)를 지원하여 행위모델의 일차적 검증을 수행할 수 있다.

5.5 적용결과

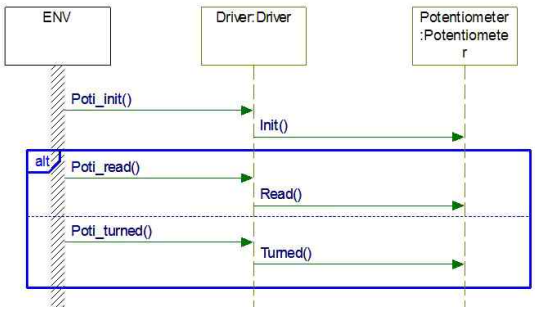
실험을 수행한 환경은 Intel CPU E8400 3.00GHz, 3GB RAM PC로 OS는 Windows 7 상에서 수행하였다. 모델 검증기 SPIN은 6.0.1버전(version)을 사용하였다.

5.5.1 자동차 거울 조정 시스템

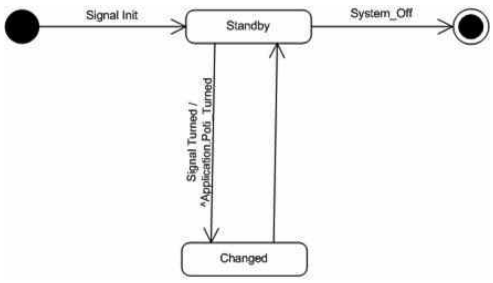
자동차 거울 조정 시스템[5]은 자동차 양쪽 사이드 거울의 위치를 자동으로 조정하는 시스템으로, 사용자 인터페이스를 제공하고 조정된 거울의 위치등을 LCD 화면으로 표현한다. 자동차 거울 조정 시스템의 컴포넌트 모델을 대상으로 모델 변환을 수행한 결과, 다음과 같이 Driver 컴포넌트와 Potentiometer 컴포넌트 사이의 행위일관성 위반을 발견할 수 있었다. (그림 5)의 구조모델과 같이 Driver 컴포넌트는 외부로 Poti_Read, Poti_turned와 같은 기능을 제공하고



(그림 5) Driver 컴포넌트와 Potentiometer의 외부구조



(그림 6) Driver의 내부행위



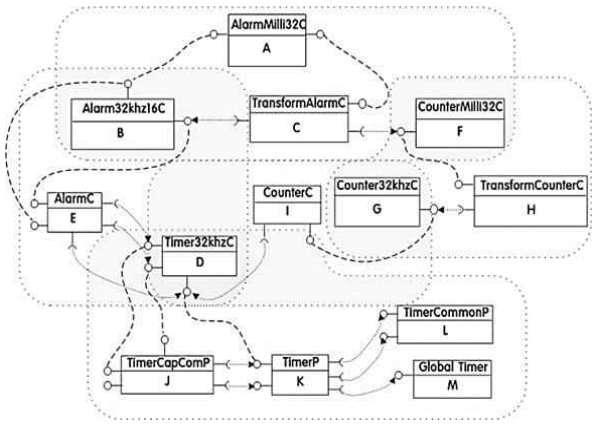
(그림 7) Potentiometer의 외부행위

해당 기능은 Potentiometer 컴포넌트에 의해 정제된다. 하지만 (그림 7)의 Potentiometer 외부행위에서 Poti_Read에 대한 전이 및 상태가 누락되어 있음을 알 수 있다. 때문에 Driver 컴포넌트가 외부로 제공하는 Poti_Read의 기능은 Potentiometer 컴포넌트로부터 제대로 정제되지 못함을 알 수 있고, 이를 모델에 반영, 수정할 수 있었다.

5.5.2 TinyOS 컴포넌트

무선센서네트워크의 OS인 TinyOS는 컴포넌트를 지향하는 nesC 언어로 설계되었다. (그림 8)은 TinyOS소스코드의 일부를 역모델하여 생성한 추상컴포넌트 구조도이다.

본 실험에서는 TinyOS의 AlarmMilli32C 컴포넌트 모델을 대상으로 모델 변환을 수행하고 행위일관성검증을 수행했다. (그림 8)에서 A, B, D, F, G는 추상 컴포넌트이고, 나머지 C, E, H, I, J, K, L, M은 물리 컴포넌트이다. (그림 8)



(그림 8) TinyOS의 AlarmMilli32C 추상컴포넌트 구조

에서 컴포넌트들을 둘러싼 테두리는 정제관계를 표현한 것으로, 예를 들어 컴포넌트 D는 J, K, L, M 컴포넌트에 의해 정제된다.

〈표 3〉 AlarmMilli32C모델의 추상 컴포넌트 검증결과

	State	*Transition*	*Depth*	*Memory usage (Mbyte)*	*Time (Second)*
A	77,097	202,304	295	52.11	6.14
B	475,517	964,907	1,672	188.04	19.50
D	33,072	73,338	136	12.66	1.18
F	677,481	1,426,165	5,206	279.26	31.00
G	506,594	1,093,790	2,573	182.38	20.30

검증은 AlarmMilli32C 모델의 추상 컴포넌트를 대상으로 수행했으며, 검증을 위해 해당 추상 컴포넌트를 정제하는 물리 컴포넌트들을 모두 추출하여 일관성모델을 구성하였다. <표 3>은 AlarmMilli32C 모델의 검증결과로, 수치들이 가지는 의미는 다음과 같다.

- State : 해당 모델을 표현하기 위해 사용된 상태의 개수
- Transition : 상태간 트랜지션(transition)의 개수
- Depth : 깊이 우선 탐색알고리즘 수행결과 루트(root)로부터 거쳐온 트랜지션의 개수
- Memory usage : 해당 컴포넌트를 검증하기 위해 사용된 메모리의 크기
- Time : 검증을 수행하는데 걸린 시간

6. 결 론

컴포넌트 기반 개발 방법은 컴포넌트의 재사용으로 인한 개발 비용의 절감뿐 아니라, 컴포넌트 분석의 재사용으로 인한 검증비용의 절감과 시스템의 질적 향상을 도모할 수 있는 장점을 지니고 있는 반면, 컴포넌트들간 상호작용의 복잡도의 증대로 인한 결함률이 높아질 수 있다.

컴포넌트간 상호작용 복잡도의 증대에도 불구하고 이를 해결하려는 다양한 연구가 존재했지만 개발방법론과의 효과적인 결합이 고려되지 않았고 검증의 기술적인 초점에만 맞춰져 일반적인 적용이 쉽지 않다는 단점과 상향식 검증을 지향하기 때문에 정제 프로세스를 갖는 하향식 개발과정에서는 적용이 힘들다는 단점이 있었다.

본 연구에서는 모델 기반 컴포넌트 개발방법 중 하나인 MARMOT의 추상 컴포넌트의 정제과정에서 추상 컴포넌트의 행위 및 환경 모델을 행위일관성 검증 모델로 변환하는 변환기를 개발하였다. 이 도구는 여러 단계에 걸쳐 발생하는 컴포넌트들의 복잡한 정제과정에서 발생 가능한 행위 일관성 문제를 조기에 찾아 낼 수 있다. 뿐만 아니라, 모든 컴포넌트를 조합하여 행위일관성을 검증할 필요 없이 각 정제단계간의 행위일관성 검증을 수행함으로써 기존의 복잡했던 반례의 분석이 비교적 쉬워졌을 뿐만 아니라 모델 검증의 복잡도를 줄이는 측면에서도 기여한다.

참 고 문 헌

[1] 김도형, 정기원 “객체지향 분석과정에서 오류와 일관성 점검 방법”, 정보과학회 논문지(B), 제26권 제3호, pp.335-451.

[2] 김치수, 진영진, “객체지향 분석의 완전성과 일관성 검증을 위한 툴의설계”, 정보처리학회 논문지, 제4권 제10호, pp.2453-2460.

[3] 김호준, 이우진, “UML 상태기계 다이어그램을 이용한 컴포넌트 인터페이스의 행위 호환성”, 정보처리학회논문지D, 제16권 제1호, pp 65-72.

[4] 최윤자, “내장형 소프트웨어 컴포넌트의 상향식 합성과 검증”, 정보처리학회논문지D, 제17권 제6호, pp.415-422.

[5] Y. Choi and C. Bunse, “Design verification in model-based μ-controller development using an abstract component”, Journal of Software Systems and Modeling, Vol.10, Issue 1, 2011.

[6] G. J. Holzmann, The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley Publishing Company, 2003.

[7] J. Hatcliff, W. Deng, M. B. Dwyer, G. Jung, V. P. Ranganath, “Cadena: an integrated development, analysis, and verification environment for component-based systems,” Proceedings of 25th International Conference on Software Engineering, pp.160-173, 2003.

[8] A. Basu, M. Bozga, and J. Sifakis, “Modeling Heterogeneous Real-Time Components in BIP,” Proceedings of 4th IEEE International Conference on Software Engineering and Formal Methods, pp.3-12, IEEE Computer Society, 2006.

장 훈

e-mail : billjang@knu.ac.kr
2008년 경북대학교 전자전기컴퓨터학부(학사)
2010년 경북대학교 전자전기컴퓨터학부 (공학석사)
2010년~현 재 LG전자 HA 제어연구소
관심분야 : 소프트웨어공학, 모델검증 등

박 민 규

e-mail : pqrk8805@hanmail.net
2011년 경북대학교 IT대학 컴퓨터학부(학사)
2011년~현 재 경북대학교 전자전기컴퓨터학부 재학(공학석사)
관심분야 : 컴포넌트기반개발, 프레임워크

최 윤 자

e-mail : yuchoi76@knu.ac.kr
1991년 연세대학교 수학과(이학사)
1993년 연세대학교 수학과(이학석사)
1993년~1996년 삼성데이터시스템
1999년 미네소타대학 전산과(이학석사)
2003년 미네소타대학 전산과(박사)
2003년~2006년 독일 프라운호퍼연구소 연구원
2006년~2008년 경북대학교 전자전기컴퓨터학부 전임강사
2008년~현 재 경북대학교 IT대학 컴퓨터학부 조교수
관심분야 : 소프트웨어 안전성 분석, 모델기반개발방법론