DOI: 10.3745/KIPSTD.2010.17D.6.415

내장형 소프트웨어 컴포넌트의 상향식 합성과 검증

최 윤 자†

요 약

본 논문은 단위 컴포넌트가 제공하는 서비스를 중심으로 컴포넌트 행위모델을 합성하고 상위수준의 추상적 행위모델을 추출하는 서비스기반 합성과 검증기법을 제안한다. 이를 위하여, 상향식 행위양식 추상화의 기초가 되는 추상 컴포넌트를 정의하고, 포트기반 동기화 알고리즘과 서비스기반 투영을 통한 추상화기법을 제안하였다. 또한, 제안된 기법을 적용한 모델검증 프레임워크를 개발하고, 사례연구를 통하여 제안된 방식이 검증비용을 절감함을 입증하였다.

키워드 : 상향식 합성, 추상화, 정형검증

Bottom-up Composition and Verification of Embedded Software

Yunja Choi†

ABSTRACT

This paper proposes service-oriented composition and verification techniques for incrementally extracting high-level abstract behavior of unit components in a systematic manner. Proposed techniques include the definition for abstract component, which is a basic building-block of the abstraction process, an algorithm for port-based synchronized abstraction, and projection abstraction. A verification framework is developed using the proposed techniques and its efficiency is demonstrated through a case example.

Keywords : Bottom-up Composition, Abstraction, Formal Verification

1. 서 론

컴포넌트기반 개발방식은 자동차, 항공, 통신 등의 중요 기간시스템에서부터 이동전화, 도난방지 시스템 등의 생활가전 등 우리 일상 생활에 널리 애용되고 있는 컴퓨터 시스템들을 제어하는 소프트웨어의 개발에 활발히 적용되고 있다. 이것은 나날이 증대되는 소프트웨어의 크기와 복잡도를 제어하기 위한 필연적인 변화라고 할 수 있다. 그러나, 현재의 컴포넌트 기반 개발방식은 코드중심의 상향식 개발방식에서 크게 탈피하지 못하고 있어, 정확도, 신뢰성, 안전성과 같은 시스템 전반적인 요구사항을 상위수준에서 검증하기 어려운 한계를 지니고 있다. 상향식 컴포넌트 조합방식은 개개의 단위 컴포넌트에 대한 이해와 검증은 용이하나, 컴포넌트가 조합되어 개발된 시스템의 전반적 행위양식에 대한 이해 부족을 야기하게 되고, 단위 컴포넌트의 구체적인 행위양식을 모두 고려한 검증방식만이 가능하여 검증비용과 효율의 저하를 초래하게 된다.

본 논문에서는 단위 컴포넌트가 제공하는 서비스를 중심으로 컴포넌트 행위모델을 합성하고 상위수준의 추상적 행위모델을 추출하는 서비스기반 합성과 검증기법을 제안한다. 행위모델의 합성은 조합되는 컴포넌트들간의 연결구조와 조합 후 제공될 서비스 정보를 기반으로, *포트기반 동기화*와 *서비스중심 투영* 방식을 적용, 상향식 추상화 과정에 순차적으로 적용된다. 이 방식은 단위 컴포넌트의 행위양식으로부터 상위 추상 컴포넌트의 행위모델을 체계적으로 추출함으로써, 단위 컴포넌트의 상위개념인 추상 컴포넌트의 명세를 가능하게 하고, 추상 컴포넌트 중심의 하향식개발과 검증의 도입을 지원하는 기초를 제공한다.

이를 위하여 본 논문은 상향식 행위양식 추상화의 기초가 되는 추상 컴포넌트를 정의하고, 포트기반 동기화 알고리즘과 서비스기반 투영을 통한 추상화기법을 제안한다. 또한, 제안된 기법을 이용한 모델검증 프레임워크를 개발하고 사례연구

※ 이 논문은 2008년-2009년 정부(교육과학기술부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2008-331-D00525).
† 정 회 원 : 경북대학교 컴퓨터학부 조교수
논문접수 : 2010년 2월 17일
수 정 일 : 1차 2010년 4월 8일, 2차 2010년 5월 6일
심사완료 : 2010년 6월 3일

를 통하여 제안된 방식이 검증비용을 절감함을 입증하였다

본 논문의 구성은 다음과 같다. 1장의 서론에 이어서, 2장과 3장에서 본 연구의 배경과 관련연구를 간략히 소개한다. 4장과 5장은 인터페이스와 포트의 역할을 강조한 컴포넌트 모델과 추상화기법을 각각 정의하고, 이 방식을 적용한 모델검증 방안을 소개한다. 6장에서 사례연구를 통해 제안된 방식의 효용성을 논하고 7장에서 연구결과를 정리한다.

2. 연구배경

본 연구는 기존의 상향식 컴포넌트 개발방식을 모델기반 하향식 개발방식에 접목하기 위한 기초작업으로써 수행되었다. (그림 1)은 모델기반, 컴포넌트 중심 개발방법론 MARMOT [8]의 추상 컴포넌트 모델을 도식화한 것이다. 이 개발방식에서는 개발대상 시스템을 하나의 추상 컴포넌트로 간주하고 세분화와 정제를 이용한 점진적인 컴포넌트의 디자인 과정을 통해 단위 물리 컴포넌트를 개발한다.

(그림 1)의 예에서, 추상 컴포넌트인 Alarm 은 32khz 알람을 구현하는 AlarmA와 카운터 기능을 수행하는 Counter 컴포넌트로 세분화될 수 있으며, 세분화 과정에서 도출된 AlarmA 는 다시 타이머의 기본 기능을 수행하는 TimerA 와 부가기능을 수행하는 TimerB로 세분화될 수 있다. 이렇게 세분화 과정에서 도출되는 컴포넌트들은 실제 구현되는 물리적인 컴포넌트와 차별화하여 추상 컴포넌트로 불린다.

(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 Counter 추상 컴포넌트의 내부모델은 TimerB와 CounterA가 어떻게 연계되어 Counter의 기능을 구현하는지 명세하여야 한다.

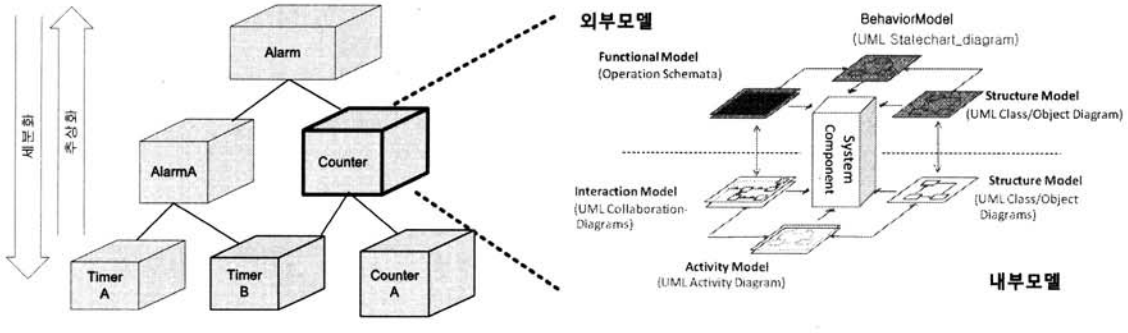
이러한 하향식 개발방식의 이점은 추상화와 세분화 수준에 따른 컴포넌트 행위모델을 관리할 수 있고, 각 세분화 과정에서 독립적인 검증이 이루어질 수 있다는 것이다. 하지만, 이러한 방식은 기존에 상향식으로 개발되어 있는 단위 컴포넌트들을 효과적으로 재활용할 수 있어야만 현실적인 적용이 가능하다. 예를 들어, 이미 개발되어 있는 TimerA, TimerB, CounterA를 재사용하여 Alarm 컴포넌트를 개발하려고 할 때, 이들 컴포넌트들의 행위양식의 조합으로 어떠한 행위양식을 도출해낼 수 있는지를 판단할 수 있어야 하며, 도출된 행위양식을 체계적으로 관리하여 점진적인 상위수준의 컴포넌트 조합과, 그 행위양식의 예측이 가능하여야 한다. 이것을 추상화과정이라 부르며, 이러한 과정을 통해 도출된 상위수준의 추상 컴포넌트는 모델기반 개발과정에서의 재사용의 단위로 활용될 수 있다.

본 연구는 이러한 추상화과정을, 이미 개발된 단위 컴포넌트의 구조모델과 조합구조를 참조하여 체계화하는 방안을 모색하였다.

3. 관련연구

본 연구에서 제안하는 컴포넌트 행위양식 조합방식은 기존의 statechart 합성기법과 유사한 점이 있다. Nejati[2]는 두 개 이상의 statechart들의 유사점을 발견하고 하나의 statechart로 합성하는 방법을 제안하였다. 이 방법은 컴포넌트와 서비스라는 개념을 도입하지 않고 일반적인 statechart를 대상으로 하여 경험적 결정이 필요하므로 기계적인 자동화가 어렵다는 문제점을 가진다. Ziadi[3]는 statechart를 합성하기 위한 대수적 프레임워크를 제안하고 UML2.0으로 작성된 순차 다이어그램으로부터 하나의 statechart 를 생성하는 방법을 제안하였다. 이 방법은 statechart 들 간의 합성보다는 순차 다이어그램의 분석과 합성에 초점을 두고 있다는 점에서 본 연구에서 제안하는 방법과 차별화된다.

행위양식을 하위단계의 모델이나 소스코드로부터 역으로 추출해낸다는 관점에서 보면, 본 연구가 제안하는 방식은 기존의 역공학적 시도의 연장선상에 있다고 볼 수 있다. 그러나, 기존의 역공학적 시도는 Java 바이트코드나 프로그램 소스코드로부터 유한상태기계나 UML 순차 다이어그램을 추출해내는 하위단계의 행위양식 추출에 국한되어 있으며 컴포넌트의 개념이 전혀 활용되고 있지 않다[4, 5]. 본 연구에서는 상위단계의 행위양식의 조합과 추상화를 추상컴포넌



(그림 1) 모델기반 컴포넌트 개발

트의 개념을 이용하여 기계적으로 적용할 수 있는 방법을 제시한다.

본 논문에서 제안하는 서비스기반 투영방식은 프로세스대수[1]의 기본 개념을 서비스 포트기반으로 응용한 것이다. 또한, 본 연구에서 제안한 추상 컴포넌트모델은 Izadi[6] 등이 정의한 컴포넌트 연결 관계를 명세하기 위한 연산모델의 확장과 응용이라 할 수 있다.

4 컴포넌트 모델

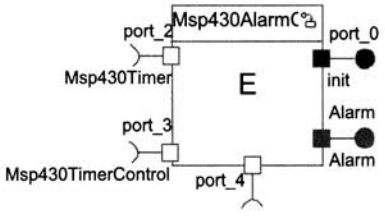
본 연구의 기반이 되는 추상컴포넌트의 개념은 Java Beans 나 EJB 와 같은 특정 물리컴포넌트와는 달리, 외부구조와 외부행위양식, 내부구조와 내부행위양식의 이원화된 블랙박스의 행태로 정의된 일반화된 개념이다.

[정의 1] 단위 컴포넌트 C=(S, R, N, P, I, E, G, A)의 구성요소들은 다음과 같이 정의된다.

- S: 컴포넌트 내부 상태들의 집합
- $R \subseteq S \times P \times E \times G \times A \times S$: 상태전이들의 집합
- N: 이름들의 집합.
- $P \subseteq N \times I \times T$: 포트들의 집합
- I: 이벤트들의 파워집합으로 정의되는 인터페이스들의 집합
- E: 컴포넌트 외부 이벤트들의 집합.
- G: 부울함수 형태의 표현식들의 집합.
- A: 단위 행위들의 집합 ■

상태간 전이는 현재 상태에서 발생한 이벤트와 그 이벤트가 전달된 포트, 전제조건에 종속적으로 미리 정의된 후속 상태로 전이되며, 이 때 전이에 따른 행위들이 수행될 수 있다. 이벤트의 종류는 시그널 또는 함수 호출 등이 있으며, 전이에 따른 행위에는 함수 호출, 시그널 전송, 변수 값 저장 등의 행위들이 가능하다. 이름들의 집합은 컴포넌트, 인터페이스, 포트 등에 유일한 이름을 부여하는 데에 사용되며, 각 포트는 이름, 인터페이스, 포트 타입 T={provide, use}의 조합으로 구성된다.

(그림 2)는 무선센서 네트워크 운영체제인 TinyOS 의 가장 하위 단위 컴포넌트를 UML2.0 표기법을 이용하여 도식화한 예이다. 이 컴포넌트의 다섯 포트들은 외부와의 모든 정보교환 및 통신의 관문 역할을 한다. 각 포트는 이름, 인터페이스, 포트타입으로 유일하게 정의된다. 예를 들어, 원형으로 표현된 포트 두 개는 각각 (port_0, init, provide), (Alarm, Alarm, provide)로 정의되었다. (port_2, Msp430Timer, use)는 반개형으로 표시된 use 포트 중 하나를 정의한 것이다.



(그림 2) 단위 컴포넌트의 표현

인터페이스는 이벤트들 또는 오퍼레이션들의 집합으로 정의되며 연계된 포트를 통해 접근할 수 있는 시그널과 함수들을 명시한다. 예를 들어, (그림 2)의 Alarm 인터페이스는 다음과 같이 정의되어 있다.

{ void start(unsigned dt), void stop(), void fired(),
bool is_Running(), void start_At(unsigned t0,
unsigned dt), unsigned getAlarm(),
unsigned getNow() }

컴포넌트 행위양식은 상태기계의 형태로 표현된다. 예를 들어, $r=(s_0$, (Alarm, Alarm, provide), start(dt), true, (Alarm, getNow()), $s_1)$은 (그림 2)에 도식화된 컴포넌트의 상태전이 중 하나를 표현한 것이다. 상태 s_0 에서 상태 s_1 으로의 전이 r 은 Alarm 인터페이스를 지원하는 provide 포트 Alarm으로부터 start(dt) 메시지를 전달받으면, 동일한 Alarm 포트를 통해 getNow 메시지를 보내고 s_1 상태로 전이한다. 이렇게 r=(s, p, e, g, a, t) 의 형식으로 표현된 전이의 다른 형태의 표현은 r: $s \xrightarrow{p?e[g]/a} t$ 이다. 행위양식에 관한 예시는 7장 사례연구에서 좀 더 자세히 다루었다.

추상 컴포넌트는 하나 이상의 단위 컴포넌트 또는 추상 컴포넌트들의 조합을 대표하는 개념적인 컴포넌트로서 다음과 같이 재귀적으로 정의하였다. 참고로, 다수의 단위 컴포넌트를 표현하기 위하여 각 단위 컴포넌트와 그 구성요소들을 첨자 i로 구분하였다.

[정의 2] 추상 컴포넌트 $M=(S,R,N,P,I,E,G,A, \{C_i\}_{i\in N}, Bmap)$은 다음과 같이 정의된다.

- S, R, N, P, G: 정의 1과 동일
- $I \subseteq U_{i\in N} I_i$: C_i 인터페이스들의 합집합의 부분집합
- $E \subseteq U_{i\in N} E_i$: C_i 이벤트들의 합집합의 부분집합
- $A \subseteq U_{i\in N} A_i$: C_i 행위들의 합집합의 부분집합
- $\{C_i\}_{i\in N}$: 유한개의 단위컴포넌트, 또는 추상컴포넌트들의 집합
- Bmap: $P \rightarrow U_{i\in N} P_i$, P의 원소인 각각의 포트를 단위 컴포넌트의 포트들 중 같은 인터페이스와 포트타입을 갖는 포트로 연결하는 바인딩함수
- 각 상태전이 $r=(s, p, e, g, a, t) \in R$에 대하여 $Bmap(p)=p_i$를 만족시키는 포트 p_i 와 상태전이 $r_i=(s_i, p_i, e, g, a_i, t_i)$가 단위컴포넌트 C_i에 존재한다. ■

추상 컴포넌트의 상태전이는 외부이벤트에 의해 유발되며,

각 외부이벤트는 추상컴포넌트를 구성하는 단위 컴포넌트들 중 하나의 상태전이를 유발하도록 추상 컴포넌트의 서비스 포트와 그 하위 컴포넌트의 서비스 포트가 바인드 되어 있을 수 있다. 예를 들어, (그림 1)의 Alarm 컴포넌트는 AlarmA와 Counter의 조합을 대표하는 추상 컴포넌트로 해석될 수 있으며, Counter와 AlarmA도 각기 독립적인 컴포넌트들의 조합으로 구현된 추상 컴포넌트로 볼 수 있다.

이러한 추상 컴포넌트의 외부행위 양식은 두 가지 방식으로 명세 가능하다. 외부행위 양식을 먼저 명세하고, AlarmA와 Counter의 조합으로 형성되는 내부행위 양식과의 일관성을 검증하는 하향식 방식과, 내부행위 양식으로부터 외부행위 양식을 추출해내는 상향식 방식이 그것들이다.

본 논문에서는, 표기상의 편리함을 위하여, 추상 컴포넌트의 내부구조와 포트연결관계가 관심사가 아닐 경우에는 추상 컴포넌트와 단위 컴포넌트의 표기를 M=(S, R, N, P, I, E, G, A)로 단일화하기로 한다.

5. 서비스 기반 행위양식의 합성 및 추출

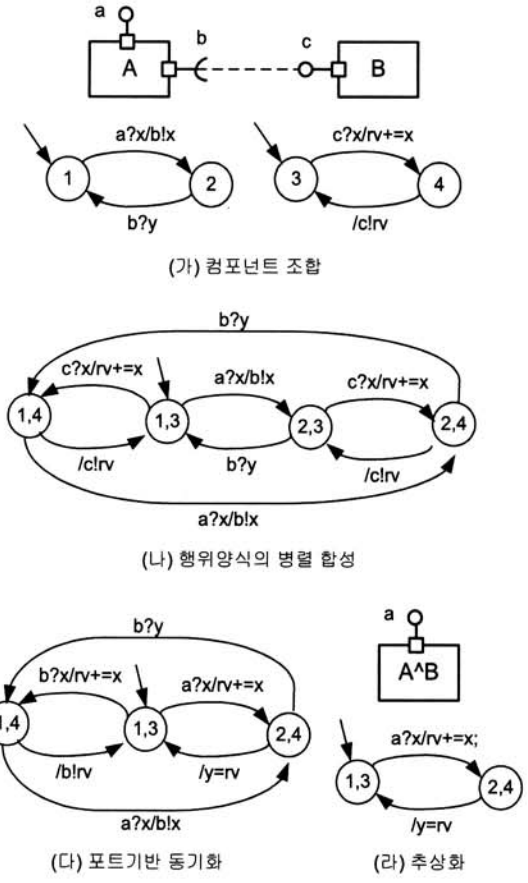
컴포넌트 행위양식의 합성은 각 단위 컴포넌트의 행위양식을 표현하는 상태기계들의 조합으로 정의될 수 있다. 하지만, 기존의 병렬조합 기법은 컴포넌트가 외부로 제공하는 서비스나 포트의 연관관계를 고려하지 않아, 기하급수적인 행위양식의 증가를 초래할 가능성이 크다. 이 장에서 제안하는 포트기반 동기화와 서비스중심 투영기법은 하위 컴포넌트들의 조합을 대표하는 상위 추상 컴포넌트의 행위양식을 포트 연결정보를 이용하여 추출하는 기법이다.

5.1 포트기반 동기화

포트기반 동기화(port-based synchronized reduction) 방식은 추상 컴포넌트의 구성에서 정의된 포트간의 결합 관계를 고려하여 합성된 행위양식을 최적화하는 것을 목적으로 한다.

예를 들어, (그림 3)의 (가)는 간단한 두 개의 독립적 추상 컴포넌트 A와 B가 포트 b와 c로 연결된 구조와, 각각의 행위양식을 상태전이기계의 형태로 표현한 것이다. 추상 컴포넌트 A는 서비스 제공포트 a와 서비스 사용포트 b를 가지고 있으며, 포트 a로부터 서비스요청을 받으면 포트 b로 그 요청을 전달하며 상태 2로 전이하고, b로부터 응답을 받으면 상태1로 전이하는 행위양식으로 명세 되었다. 컴포넌트 B는 서비스 제공포트 c만을 가지며, c로부터 요청이 들어오면 그 요청메시지를 합산하고 상태 4로 전이하고, 합산한 결과를 c를 통해 전달하며 상태 3으로 전이하는 행위양식을 갖는다.

이 두 컴포넌트들의 행위양식을 상태전이기계간의 합성으로만 간주할 경우, (나)와 같이 모든 가능한 상태들과 전이들의 조합으로 표현될 것이며, 만약 A와 B가 n개와 m개의 상태들로 이루어져 있다면, 조합의 결과는 최악의 경우 n×m 개의 상태들로 구성될 수 있다. 이러한 복잡도는 상위수준의 행위양식을 직관적으로 이해하는 데에 걸림돌이 될 뿐 아니라, 행위정당성 검증에도 많은 비용이 소요되는 문제점을 갖는다.



(그림 3) 포트기반 동기화

포트기반 동기화 방식은 연결된 포트들의 이름을 같은 이름으로 대치하여 메시지 전달의 구체적인 전이과정을 생략함으로써 추상 컴포넌트의 상태전이 행위양식을 추상화하는 방식이다. (그림 3)의 (다)는 컴포넌트 A 의 포트 b 와 컴포넌트 B 의 포트 c 의 연결 관계를 이용하여 (그림 3)(나)의 상태전이도에서 c를 b로 대체하고 불필요한 전이과정을 생략한 결과이다. 예를 들어, 두 개의 연속적인 전이 (1,3) $\xrightarrow{a?x/b!x}$ (2,3) $\xrightarrow{c?x/rv+=x}$ (2,4) 는 c를 b로 대체함으로써 (1,3) $\xrightarrow{a?x/b!x}$ (2,3) $\xrightarrow{b?x/rv+=x}$ (2,4) 로 바뀌고, 시간차와 버퍼링을 고려하지 않는다면 b로 메시지 x를 보내는 b!x 행위가 b에서 메시지 x를 받는 행위 b?x와 동기화되므로 중간전이가 생략된 (1,3) $\xrightarrow{a?x/rv+=x}$ (2,4) 형태로 상태전이가 단순화된다.

(그림 3)의 (라)는 두 개의 컴포넌트가 조합되어 하나의 추상컴포넌트 A^B를 구성한 구조모델과 서비스기반으로 추상화된 그 행위모델을 도식화하고 있다. A^B 추상컴포넌트가 외부에 제공하는 서비스는 포트 a에 의해서만 접근이 가능하

므로, 합성에 의해 내부포트로 변환된 포트 b와 포트 c에 의해 다루어지는 서비스들은 모두 생략되었음을 알 수 있다. 이와 같이, 컴포넌트 조합시의 포트기반 행위양식 추상화는 행위양식의 병렬조합으로부터 불필요한 행위양식을 제거해 나가는 과정을 거쳐서 완성된다.

[정의 3] 두 추상 컴포넌트 M=(S, R, N, P, I, E, G, A)와 M′=(S′, R′, N′, P′, I′, E′, G′, A′) 의 병렬조합 M|M′=($\overline{S}$, $\overline{R}$, $\overline{N}$, $\overline{P}$, $\overline{I}$, $\overline{E}$, $\overline{G}$, $\overline{A}$) 은 다음과 같이 정의된다.

- $\overline{S} \subseteq S \times S'$, $\overline{N} \subseteq N \times N'$, $\overline{P} \subseteq P \times P'$, $\overline{I} \subseteq I \times I'$
  $\overline{E} \subseteq E \times E'$, $\overline{G} \subseteq G \times G'$, $\overline{A} \subseteq A \times A'$
- $\overline{R} \subseteq R \times R'$이고 다음의 규칙을 따른다.

$$\surd \ \frac{s \xrightarrow{p?e[g]/a} t \text{ in } R,\ s' \xrightarrow{p'?e'[g']/a'} t' \text{ in } R'}{(s, s') \xrightarrow{p?e[g]/a} (t, s') \text{ in } \overline{R}}$$

$$\surd \ \frac{s \xrightarrow{p?e[g]/a} t \text{ in } R,\ s' \xrightarrow{p'?e'[g']/a'} t' \text{ in } R'}{(s, s') \xrightarrow{p'?e'[g']/a'} (s, t') \text{ in } \overline{R}}$$

■

〈표 1〉 포트기반 동기화 알고리즘

입력
√ 주어진 두 추상 컴포넌트 M 과 M′ 의 병렬조합으로 이루어진 조합컴포넌트 M|M′=($\overline{S}$, $\overline{R}$, $\overline{N}$, $\overline{P}$, $\overline{I}$, $\overline{E}$, $\overline{G}$, $\overline{A}$)
√ 같은 타입의 인터페이스를 지원하는 use 포트 p∈P 와 provide 포트 p′∈P′

출력
√ 포트기반 동기화에 의한 추상컴포넌트 $M_{(p \rightarrow p')}$ ($\dot{S}$, $\dot{R}$, $\overline{N}$, $\dot{P}$, $\overline{I}$, $\overline{E}$, $\overline{G}$, $\overline{A}$)

초기상태
√ ∀s∈$\overline{S}$, s∈$\dot{S}$ (병렬조합의 모든 상태는 추상컴포넌트의 상태에 속한다)
√ $\dot{R}$= ∅ (초기에는 상태 전이가 정의되지 않음)
√ $\dot{P}$= $\overline{P}$ ~ {p, p′} (연결된 포트들을 제외한 포트들의 집합)

알고리즘
step 1. $\overline{R}$에 연속적으로 정의된 상태전이들
$r_i : s_i \xrightarrow{q?e[g]/a} s_{i+1}$과 $r_{i+1} : s_{i+1} \xrightarrow{q'?e'[g']/a} s_{i+2}$
에 대하여,
① a= p′!x, x=e′ 이고 q′= p′ 이면 상태전이
$r_i' : s_i \xrightarrow{q?e[g \wedge g']/a'} s_{i+2}$ 을 $\dot{R}$ 에 추가
② 그 외의 경우, 각 전이 r_i, r_{i+1}을 $\dot{R}$ 에 추가
③ 추가된 전이가 있으면 step1 으로 반복, 그렇지 않으면 step2 수행
step 2. $\dot{R}$ 에 속한 상태전이 중 p, p′ 을 이용하는 전이들 삭제하고 step3 수행
step 3. 도달 전이(incoming transition)가 없는 상태 s 들을 모두 $\dot{S}$ 에서 제거.

[정의 3]은 두 추상 컴포넌트들 간의 병렬조합에 대한 연산모델을 정의한 것이다. 병렬조합으로 생성된 추상 컴포넌트 M|M′의 상태집합은 M의 상태집합과 M′의 상태집합간의 데카르트곱으로 정의되며, 전이규칙은 두 컴포넌트의 M에 속한 포트로 외부이벤트가 전달되었을 경우에는 이에 대응하는 M 에 속한 상태들 간의 전이가 발생하고, M′에 속한 포트로 외부이벤트가 전달되었을 경우에는 이에 대응하는 M′에 속한 상태들간의 전이가 발생한다. 단, 두 개의 포트에 동시에 이벤트가 전달되는 경우에도 연산은 순차적으로 수행되는 것으로 한다. 병렬조합의 결과는 상위 추상컴포넌트의 내부행위양식을 나타낸다.

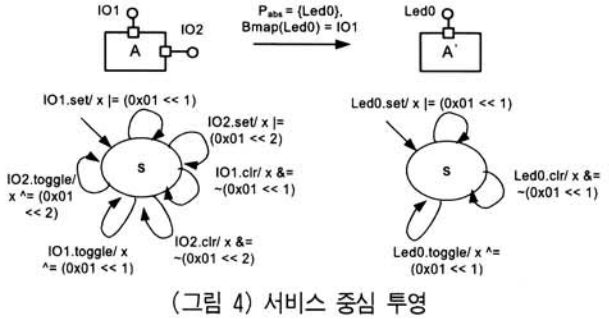
이러한 병렬조합으로부터 포트기반 동기화를 수행하여 행위양식을 추상화하는 알고리즘은 <표 1>과 같다. 다음 절에서 소개하는 서비스중심 투영 방식과 함께 적용되어, 추상화의 결과는 상위 추상 컴포넌트의 외부행위 양식을 나타낸다.

5.2 서비스 중심 투영

컴포넌트의 추상화는 두 개 이상의 컴포넌트의 조합뿐 아니라 단일 컴포넌트에서도 발생할 수 있다. 대부분의 내장형 소프트웨어 컴포넌트들은 다양한 조합을 염두에 두고 여러 기능들을 제공하지만, 특정 응용 프로그램에서 사용되는 것은 그 중의 몇 가지 기능에 불과하다. 이러한 경우, 해당 컴포넌트의 전체 행위양식을 사용되는 서비스중심으로 추상화하여 단순화된 행위모델을 추출해낼 수 있다.

(그림 4)는 서비스중심 투영을 통한 컴포넌트 행위양식의 추상화의 간단한 예를 도식화한 것이다. 컴포넌트 A 는 같은 인터페이스를 지원하는 두 개의 서비스 포트를 제공한다. 즉, 두 서비스 포트 IO1과 IO2는 모두 {set, toggle, clr}의 서비스를 지원하지만 A를 사용하는 응용 프로그램은 하나의 포트에서 제공하는 서비스만을 사용할 수 있다. 이때, A 를 하나의 서비스포트만을 지원하는 컴포넌트로 추상화하여 행위양식을 단순화하는 것이 서비스중심 투영방식이다. [정의 4]는 서비스중심 투영을 추상 컴포넌트의 개념에 적용하여 정의하였다.

[정의 4] 주어진 추상컴포넌트 M=(S, R, N, P, I, E, G, A)과 포트들의 집합 P_{abs}, 그리고 포트 바인딩 맵 BMap: P_{abs} →P에 대하여, 다음과 같이 정의되는 컴포넌트 M′ =(($\overline{S}$, $\overline{R}$, $\overline{N}$, P_{abs}, $\overline{I}$, $\overline{E}$, G, A)은



(그림 4) 서비스 중심 투영

P_{abs}에서 지원하는 외부 이벤트만을 처리하는 추상컴포넌트이다.

- $\overline{N}$은 P_{abs}에 정의된 포트 이름들의 집합
- $\overline{I}=\{i\in I \mid \exists p\in P_{abs},\ n\in\overline{N},\ t\in T\ \ni\ p=(n, i, t)\}$, 즉, $\overline{I}$은 추상컴포넌트가 제공하는 포트들에 의해 지원되는 인터페이스들의 집합
- $\overline{E}=\{e\in E \mid \exists p\in P_{abs},\ n\in\overline{N},\ t\in T\ \ni\ p=(n, i, t)\wedge e\in i\}$, 즉, M의 이벤트들 중에서, M′에 의해 지원되는 이벤트들의 집합
- $\overline{S}=\{s,\ t\in S \mid r=(s, p, e, a, t)\in R,\ \ni\ p\in BMap(P_{abs})\wedge e\in\overline{E}\}$, 즉, R에 정의된 모든 상태전이들의 소스상태와 타겟 상태들의 집합
- $\overline{R}=\{r\in R \mid r=(s, p, e, a, t)\ \ni p\in BMap(P_{abs})\wedge e\in\overline{E}\}$, 즉, R에 정의된 모든 상태전이들 중 그 상태전이 촉발이벤트가 P_{abs}에 의해 지원되는 상태전이들의 집합 ■

포트기반 동기화와 서비스중심 투영을 이용해 조합된 행위양식은 조합의 결과로 정의된 추상 컴포넌트의 기능중심 외부행위 양식을 나타낸다.

5.3 추상화의 정당성

앞 절에서 소개한 포트기반 동기화와 서비스기반 투영방식은 컴포넌트간의 메시지 전달방식, 버퍼링 등의 상세한 내용을 생략하고 컴포넌트를 기능중심으로 추상화한다. 이러한 추상화방식의 정당성은 두 가지 관점에서 보장되어야 한다. 첫째, 추상화 이전의 조합 컴포넌트의 행위양식에서 메시지 전달과정에서의 문제점이 없음을 먼저 보임으로써 기능중심 추상화의 부작용이 없음을 입증하여야 한다. 둘째, 추상화 이전의 행위양식이 외부 서비스의 기능적 관점에서 추상화된 행위양식에 포함된다는 것을 보여야 한다.

추상화 이전의 조합적 행위양식에 있어서의 메시지 전달방식의 정당성은 다음 절에 소개하는 *행위일관성 검증* 방식으로 모델검증을 실시하여 입증한다. 추상화된 행위양식이 기능적인 관점에서 추상화 이전의 행위양식을 내포하고 있음은 다음의 정리에 의해 입증된다.

[정리 1] 컴포넌트 M=(S, R, N, P, I, E, G, A)이 컴포넌트 M′=(S′, R′, N′, P′, I′, E′, G′, A′)의 포트기반 동기화를 통한 추상컴포넌트이면, M′은 P의 관점에서 M의 행위정제(trace refinement)이다. ■

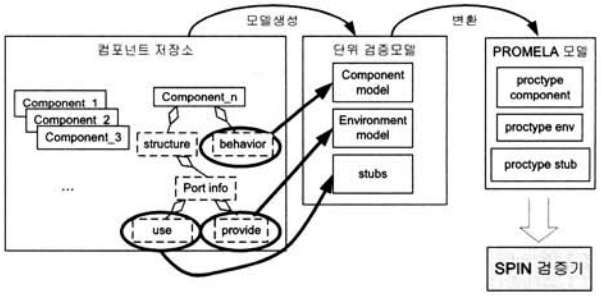
[정리 1]의 의미는, 포트기반 동기화를 통해 추출된 추상컴포넌트의 행위모델에는 본래의 행위모델의 상세한 부분이 생략되었으나, 포트 P에서 제공하는 서비스와 연관된 행위양식의 관점에서는 차이가 없음을 말한다. [정리 1]은 포트 P로 전달되는 외부 이벤트에 의해 유발되는 M의 모든 상태전이들의 순열은 M′의 상태전이들의 순열에 의해 구체화됨을 보임으로써 증명할 수 있다. 증명과정은 지면관계상 생략한다.

제안된 상향식 합성과 추상화기법으로 추출된 추상컴포넌트 행위양식은 두 단계로 나뉘어 검증된다. 첫째, 조합된 컴포넌트들 간 상호작용에 오류가 없음을 보이는 행위일관성 검증과, 둘째, 추상화된 컴포넌트 행위모델의 기능적 정확성 검증이 그것이다.

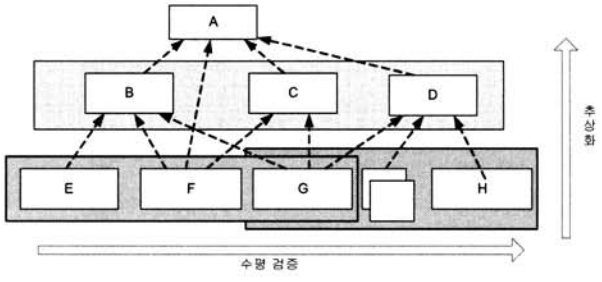
(그림 5)는 본 연구에서 제안하는 행위일관성 검증을 위한 모델변환 방식을 도식화하고 있다. 추상 컴포넌트가 제공하는 서비스에 관한 정보는 외부에 노출된 provide 포트로부터 추출되며, 이 정보는 추상컴포넌트의 외부환경 모델을 구성하는데 사용된다. 추상 컴포넌트 내부모델의 행위양식은 추상 컴포넌트를 구성하고 있는 각 컴포넌트들의 행위양식을 유한상태기계 형태로 구성하여 독립적인 프로세스들의 조합으로 표현한다. 생성된 환경모델과 행위모델은 모델검증기 SPIN[7] 의 입력언어인 PROMELA로 변환되어 정형적으로 검증된다. SPIN은 다음과 같은 추상컴포넌트의 행위일관성 검증에 사용된다.

- √ 임의의, 일련의 서비스 호출에 대하여 추상컴포넌트가 정지함 없이 서비스들을 처리한다(프로세스 교착의 부재)
- √ 서비스 호출 x 에 대하여 언젠가는 반드시 행위 y를 수행한다(서비스 요구사항의 만족)

(그림 6)은 (그림 5)의 모델변환과 검증 방식의 추상화 과정과의 연관관계를 보이고 있다. 최하위의 물리적 컴포넌트들의 조합으로부터 시작하여, 같은 추상화 수준의 컴포넌트들의 조합이 수평적으로 검증된다. 이 컴포넌트들은 각기 다시 추상화 과정을 거쳐 상위수준의 컴포넌트 구조와 행위양식으로 정의되고, 새로운 수평적 검증의 대상이 된다.



(그림 5) 모델검증 프레임워크



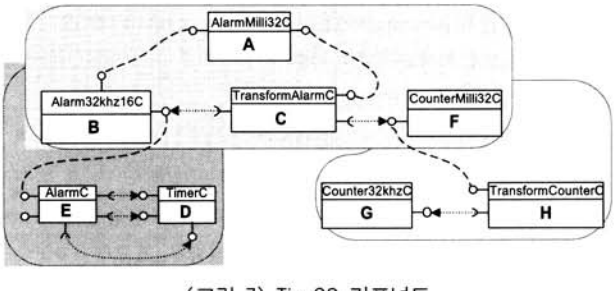
(그림 6) 추상화와 모델검증

6. 사례연구

(그림 7)은 무선센서 네트워크의 운영체제로 사용되고 있는 TinyOS의 컴포넌트 구조의 일부를 본 연구에서 제시한 추상 컴포넌트의 개념을 적용하여 도식화 한 것이다. AlarmMilli32C(A) 로 대표되는 이 추상 컴포넌트는 TinyOS의 알람 기능을 담당한다. 컴포넌트 A는 B와 C 의 서비스를 외부에 제공하는 매개 역할을 하며, 이 중 B는 다시 E 에서 제공하는 서비스를 매개하는 역할을 한다. 또한 E 는 D에서 제공하는 서비스를 이용하여 구현된다.

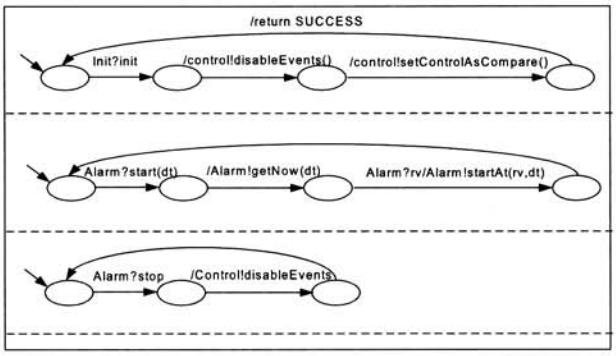
예를 들어, 컴포넌트 E, AlarmC에 해당하는 실제 프로그램 코드와 그에 해당하는 내부행위 모델은 (그림 8)과 같다. 여기서 AlarmC 가 제공하는 모든 서비스들은 외부 컴포넌트의 서비스를 사용하여 구현되었으며, 어떤 외부 컴포넌트와 연결되느냐에 따라 서비스의 내용이 달라질 수 있음을 알 수 있다. 이 예제에서는 (그림 7)에서와 같이 E 가 D 의 provide 서비스를 이용하여 구현되었으므로, 조합적 검증과 추상화 또한 이 연관관계를 이용하여 수행된다. 이 중 가장 간단한 행위양식을 보이는 stop 서비스를 예로 살펴보면, AlarmC 가 alarm 포트를 통해 전달받은 stop 요청의 결과로 control 포트를 통해 disableEvents 이벤트를 컴포넌트 D로 전달한다. D 내부의 disableEvents 요청에 대한 행위는 하드웨어 종속적인 volatile 메모리 값을 특정 값으로 초기화 하는 코드 CLR_FLAG(TxCCTLx, CCIE)로 구현되어 있다. 포트기반 동기화에 의해 E와 D가 컴포넌트 B로 추상화 되는 과정에서 E의 상태전이 $s_1 \xrightarrow{\text{alarm?stop/control!disableEvents}} s_2$와 D의 상태전이 $t_1 \xrightarrow{\text{control?disableEvents/CLR_FLAG(..)}} t_2$는 $(s_1, t_1) \xrightarrow{\text{alarm?stop/CLR_FLAG(..)}} (s_2, t_2)$로 함축된다. 이 과정을 거친 후의 B 의 행위모델은 (그림 9)와 같이 표현될 수 있다.

좀 더 복잡한 형태로 구현된 getNow 서비스에 대한 추상화 과정은 지면관계상 생략하였으나, 같은 과정을 거쳐 외부 컴포넌트에 독립적인 B의 내부행위 양식을 도출해낼 수 있다. 이 추상화 과정에서 특정 하드웨어 메모리 참조와 수정은 전역변수의 참조와 수정으로 대체하여 하드웨어 비종속적인

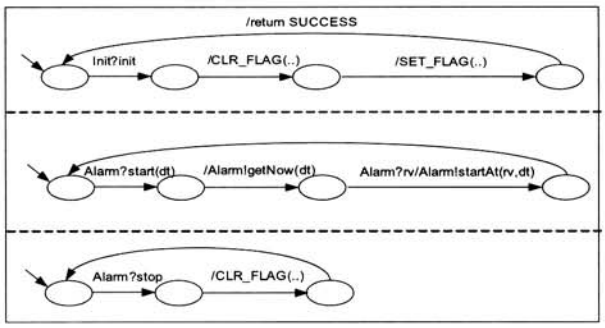


(그림 7) TinyOS 컴포넌트

```
generic module AlarmC(..){
  provides interface Init;
  provides interface Alarm<..> as Alarm;
  uses interface Timer;
  uses interface TimerControl;
  uses interface Compare;
}
implementation{
  command error_t Init.init(){
    call Control.disableEvents();
    call Control.setControlAsCompare();
    return SUCCESS;
  }
  async command void Alarm.start(uint16_t dt){
    call Alarm.startAt(call Alarm.getNow(), dt);
  }
  async command void Alarm.stop(){
    call Control.disableEvents();
    signal Alarm.fired();
  }
  ...
}
```



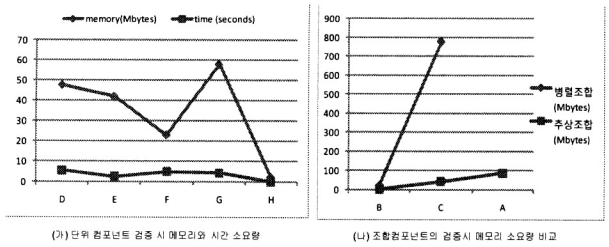
(그림 8) TinyOS 코드와 행위모델 예시



(그림 9) 포트기반 추상화 예시

모델로 변환하였다.

(그림 10)은 (그림 7)에 예시된 TinyOS 추상 컴포넌트들의 모델 검증 시 소요되는 메모리와 시간을 비교한 그래프이다. (그림 10)(가)는 컴포넌트A 를 구성하는 가장 밑단의 단위 컴포넌트들을 프로세스 교착상태의 부재에 관하여 모델 검증한 결과를 메모리와 시간 소요량의 측면에서 분석한 것이다. 단위컴포넌트의 크기에 따라 변화를 보이지만, 60Mbyte 의 시스템 메모리와 10초 이내의 시간 안에 SPIN검증기를 이용하여 프로세스 교착상태의 부재가 검증되었음을 볼 수 있다. (그림 10)의 (나)는 이들 단위 컴포넌트들의 조합으로 구성된 추상 컴포넌트들의 행위일관성 검증의 결과를 메모리 소요량을 기준으로 추상화 전, 후로 비교한 것이다. 병렬조합만으로 행위양식이 구성되었을 때에는 B, C, A의 순서로 복잡도가 증가됨에 따라 검증에 소요되는 메모리의 양이 급격히 늘어

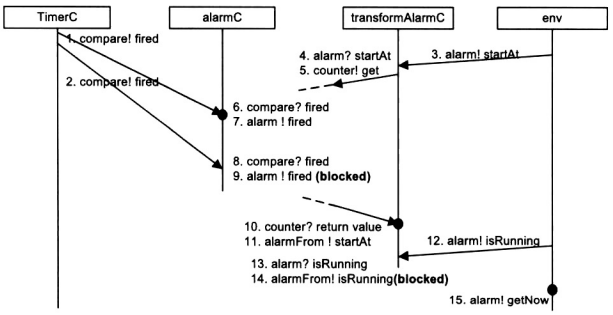


(가) 단위 컴포넌트 검증 시 메모리와 시간 소요량 (나) 조합컴포넌트의 검증시 메모리 소요량 비교

(그림 10) 검증효율 비교

남을 볼 수 있는 반면, 병렬조합 후 서비스 기반 추상화 기법을 적용하였을 경우는 그 증가 속도가 현저히 둔화됨을 볼 수 있다. 특히, 추상 컴포넌트 A의 경우에는 병렬조합으로 컴포넌트 행위양식을 표현하였을 경우 1Gbyte의 메모리 한계 내에서 검증에 성공하지 못한 반면, 추상화 후에는 100Mbytes 이내에 검증을 완료하였다.

(그림 11)은 검증과정에서 SPIN이 발견한 잠재적인 위험요소들 중 하나를 도식화 한 것이다. 외부에서 요청된 startAt 서비스 수행 중 타임아웃을 알리는 이벤트 fired가 두 번 연속 발생하였을 경우, 이 이벤트들은 alarm 포트의 버퍼에 블로킹될 가능성이 있다. 이것은 TransformAlarmC가 startAt 서비스 수행이 끝나지 않은 상태에서 fired 이벤트를 처리할 수 없을 경우에 발생할 수 있는 상황이다. 이렇게 alarm 포트가 블로킹된 이후에 발생하는 외부 서비스 요청은 전체 프로세스들을 교착상태에 이르게 할 가능성이 있음을 나타낸다. 이러한 반례들의 분석을 통해 잠재적인 위험요소들을 식별해냄으로써 모델의 안전성과 신뢰성을 높이는데 기여할 수 있다.



(그림 11) 검증반례 예시

7. 결 론

본 논문에서 제안한 서비스기반 컴포넌트 합성기법은 내장형 소프트웨어의 개발과 검증과정에 두 가지 관점에서 기여할 수 있다. 첫째, 상위수준의 컴포넌트 모델을 정의하고 상향식 합성을 통해 행위모델을 기계적으로 도출할 수 있는 방법을 제시하여, 컴포넌트의 재사용과 검증의 단위를 상위수준으로 높이는 데에 사용될 수 있다. 둘째, 체계적인 합성과 검증기법의 적용으로 검증 효율을 높임으로써, 상위수준 컴포넌트의 신뢰성과 안전성을 높일 수 있다.

본 연구의 접근방식이 검증의 효율을 높일 수 있다는 것은 예제를 통하여 입증되었다. 그러나, (그림 10)에서 나타나듯이, 검증에 소요되는 메모리와 자원은 추상화 후에도 복잡도의 증가에 따라 증가추세에 있음을 볼 수 있다. 이것이 기하급수적인 증가가 아닌 선형증가가 됨을 이론적으로나 실험적으로 입증하는 것이 본 연구의 다음 과제이다.

참 고 문 헌

[1] C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.

[2] Shiva Nejati et al., Matching and merging of statecharts specifications, In 29th IEEE International Conference on Software Engineering, pp.54-64, 2007.

[3] Tewfik Ziadi, Loeic Heloueut, and Jean-Marc Jezequel. Revisiting statechart synthesis with an algebraic approach. In 26th IEEE International Conference on Software Engineering, pp.242-251, 2004.

[4] Lionel C. Briand, Yvan Labiche, and Johanne Leduc. Toward the reverse engineering of UML sequence diagrams for distributed Java software. IEEE Transactions on Software Engineering, Vol.32, No.9, pp.642-663, July/August 2006.

[5] Y.Yu, Y.Wang, J.Mylopoulos, S. Liaskos, A. Lapouchnian, and J. Leite. Reverse Engineering goal models form legacy code. In 13th IEEE International Requirements Engineering Conference, pp.363-372, 2005.

[6] Mohammad Izadi, Marcello M. Bonsangue, and Dave Clarke. Modeling Component connectors: Synchronization and Context-dependency. In 6th IEEE International Conference on Software Engineering and Formal Methods, pp.303-312, 2008.

[7] Gerard J. Holzmann, The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley, 2003.

[8] Yunja Choi and Christian Bunse, Design Verification in Model-based micro-Controller Development using an Abstract Component, Software and Systems Modeling, to appear.

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 프라운호퍼연구소 연구원
2006년~2008년 경북대학교 전자전기컴퓨터학부 전임강사
2008년~현 재 경북대학교 IT대학 컴퓨터학부 조교수
관심분야 : 소프트웨어 안전성 분석, 모델기반개발방법론