컴포넌트 기반 개발에서 처리 부품들의 합성을 검증하는 방안

주 복 규† · 김 영 철†

요 약

컴포넌트 기반 소프트웨어 개발 방법은 점증하는 소프트웨어의 복잡성에 대처하고, 생산성을 높일 수 있는 가장 주요한 수단의 하나로 인식되고 있다. 이 논문은 자료흐름 시스템 구조에 따른 설계서로 부품을 합성하여 소프트웨어를 개발하는 경우에, 두 부품의 영역관계를 분석하였다. 그 결과로 조립 시에 개별 부품들의 시험 자료를 이용하여 통합을 위한 시험 자료를 생성하는 방안을 제시하였으며, 이는 통합을 효과적으로 검증할 수 있게 해 준다. 또한 실험을 통해서 제안된 기법의 실제 적용 가능성을 보였다. 이 연구에서 사용한 영역 분석은 영역 시험 기법을 응용한 것이다.

A Validation Technique for Process Component Composition in the Component-Based Development Process

Bok-Gyu Joo† · Young-Chul Kim†

ABSTRACT

Component-based development methods are regarded as one of the most important tools for us to cope with ever-increasing software complexity and, at the same time, to improve productivity in software development. This paper presents how to generate test-cases for integrated software from those of member components and how to validate composition of components, by the analysis of domain relations of components. This paper shows the applicability of the proposed technique by the real experiment. This study is based on the dataflow systems architecture and process components, and the technique developed here is an application of domain testing technique.

키워드 : 컴포넌트 기반 개발 방법(component-based development process), 소프트웨어 통합 시험(software integration testing), 시험 케이스 생성(test-case generation), 영역 시험 기법(domain testing technique), 영역 분석 기법(domain analysis technique)

1. 서 론

현대 사회에서 모든 사회활동이 컴퓨터에 의존하고 있으며 이에 따라 소프트웨어의 양이나 그 복잡도는 계속 증가하고 있다. 이는 하드웨어를 컴퓨터 소프트웨어를 중심으로 한 시스템의 하나의 부품으로 변하게 하고 있으며 또한 소프트웨어에 요구되는 사항은 점점 더 크고 복잡해지고 있다.

그 동안 소프트웨어 개발 방법은 많은 발전을 했다. 대표적인 발전 분야는 재사용 방법론, 객체지향 방법론, 그리고 최근의 부품 기반 개발 방법론이 그것이다. 재사용 방법론은 기 개발된 소프트웨어 산출물을 새로 만들어지는 시스템의 개발에 다시 사용하는 것으로, 점증하는 소프트웨어에 대한 요구를 대처할 수 있는 중요한 수단으로 인식되고 있다. 객체지향 개념의 등장과 그에 따른 프로그래밍 언어의 발전으로 우리가 지금까지 자연스럽게 표현하지 못하였던 새로운 시스템을 모델링하고 설계할 수 있게 되었다.

지금은 컴포넌트 기반 개발(CBD : Component-Based Development) 방법으로 재사용 방법과 객체 지향 방법에 관한 연구 결과들이 통합 발전되고 있다. 이미 개발되어 품질이 검증된 부품들을 조합해서 새로운 시스템 개발에 사용함으로써 개발 기간을 단축하고, 소프트웨어의 품질을 높이는데 획기적으로 기여할 것으로 믿어지고 있다[12,13].

CBD 방법에서는 효과적으로 필요한 부품들을 찾아서 통합하는 것이 새롭게 해결해야 할 문제이다. 특히 다른 목적으로 개발된 부품들을 설계서에 따라 통합할 때 그 검증 방법은, 자체 개발을 염두에 두고 설계하고 개발 한 경우보다, 더 확실하고 엄격해야 한다. 전체 개발 기간의 40%가 넘는 기간이 검증에 소요되고 있는 현실에서, 통합 결과를 효율적으로 검증하는 것과 좋은 시험 자료를 만드는 방법은 특히 중요하다.

† 정 회 원 : 홍익대학교 전자전기컴퓨터공학부 교수
논문접수 : 2001년 10월 5일, 심사완료 : 2001년 11월 8일

이 논문에서는, 자료흐름 시스템(dataflow systems) 구조에서 부품들의 영역 관계를 분석함으로써 처리 부품[1]을 통합할 때 개별 부품의 시험 자료를 이용하여 통합 시험에 사용할 시험 자료로 만드는 방법과 통합 시스템의 시험 케이스의 효율성을 검증하는 방법을 제시한다.

본 논문의 구성은, 2절에서는 통합과 시험 방법에 관련된 연구를 소개하고, 3절에서는 처리 부품의 정의와 그들의 통합 방법을 설명하고, 4절에서는 영역 분석 방법에 의해 개별 부품의 시험 자료를 이용하여 어떻게 통합 부품의 시험을 설계하고 검증하는 가를 보여준다. 5절에서는 실험결과를 설명하고, 6절에는 연구의 결과를 정리하면서 향후 연구 방향을 제시 한다.

2. 소프트웨어 통합과 통합시험에 관한 연구

컴포넌트 기반 개발 방법에서의 개발 단계는 기존의 개발 단계 중 단위 모듈의 개발과 시험 대신, 필요한 부품을 찾는 단계가 추가되고, 설계와 통합 단계도 약간의 변화가 있다. 이러한 환경에서의 개발 과정은 다음과 같이 요약할 수 있다.

단계 1 : 요구 분석
단계 2 : 소프트웨어 구조 설계. 이때 활용 가능한 부품을 고려하며 진행한다.
단계 3 : 필요한 부품을 찾기.
단계 4 : 부품 통합과 시험. 부품 접합(adaptation)과 개발을 포함한다.
단계 5 : 시스템 시험

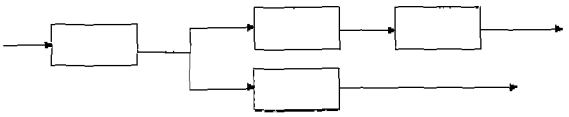
기 개발된 부품을 이용하여 새로운 시스템을 개발함으로써 개발 생산성과 품질을 획기적으로 높일 수 있다는 매력적인 생각은 실제 적용에서 해결해야 할 여러 가지 문제를 가지고 있다. 기술적인 측면에서는 부품 저장소의 구성, 필요한 부품을 찾는 문제[15], 프레임웍 문제 등이 있다. 이러한 기술 적인 문제들의 핵심에 있는 것이 소프트웨어 구조의 문제이다[3, 14].

시스템 구조와 처리 부품 : 소프트웨어 구조(software architecture)의 선택은 전체 개발에서 매우 중요한 요소이다. 특히 CBD 환경에서는, 어떤 종류의 컴포넌트 모델을 기대하며 어떤 구조 설계서를 작성하고 어떤 방법으로 통합할 것인가 등에 심대한 영향을 미친다. Shaw와 Garlan의 저서[2]에서는 현존하는 소프트웨어 시스템의 구조를 크게 다섯 가지로 분류하였다. 즉,

- Dataflow systems
- Call-and-return systems
- Independent components
- Virtual machines
- Data-centered systems (repositories)

본 연구는 자료흐름 시스템(dataflow systems) 구조와 그에 따른 부품 모델을 대상으로 하였다. 이 구조에서 각 부품은 입력 집합과 출력 집합이 있으며, 부품은 자료의 스트림을 읽어서 출력으로 내보낸다. 그 내부에서는 입력 스트림에 대하여 부수 효과(side-effect)가 없는 변환을 행하고, 그 변환은 순차적으로 이루어지며, 출력은 입력이 다 들어오기 전에 시작된다.

이 구조의 가장 중요한 특징은 두 가지 이다. 하나는 각 부품은 독립된 개체로서 다른 부품과 상태를 공유하지 않는다는 것이고, 다른 하나는, 각 부품은 입력의 원천과 출력의 목적지를 알지 못하는 상태로 동작한다는 것이다. (그림 1)은 이러한 구조를 보여준다.



(그림 1) 자료흐름 시스템 구조

이러한 구조의 시스템은 재사용 및 CBD 관점에서 매우 좋은 특성을 가지고 있다. 시스템 관점에서 보면,

(1) 설계자가 전체 시스템의 입출력과 행동양식을 개별 부품의 행동양식의 합성(composition)으로 파악할 수 있다.
(2) 재사용을 간단히 지원한다. 즉 부품들 사이에 주고받는 자료에 대해서 동의하면 쉽게 연결할 수 있다.
(3) 각 부품을 쉽게 바꿀 수 있어, 시스템의 유지 보수와 기능 향상이 쉽다.

또한 개발자의 관점에서 보면, 외부와의 인터페이스가 간결하고 통합과정도 다른 구조보다 명료하게 검증할 수 있다. 이 모델은 대부분의 절차적 언어에 있는 부 함수 호출뿐만 아니라, Unix의 프로세스 사이의 통합, 그리고 DFD (data-flow diagram)로 설계할 수 있는 프로세스(변환자)를 연결한 설계서의 경우에도 적용할 수 있다.

통합 시험 : 통합 시험이란 두 개의 기능적으로 검증된 부품들을 통합할 때 그 인터페이스와 상호작용을 시험하는 것이다. 검증된 부품을 통합할 경우에도 오류의 가능성은 있다[8]. 부품은 단위 시험과 기사용을 통해서 검증되었으며, 시험 자료를 가지고 있다.

지금까지 소프트웨어 시험에 관한 많은 연구가 있었다. 그럼에도 불구하고 현실에서 신뢰성 있는 시스템을 만들기 위한 소프트웨어의 시험에는 많은 어려움이 있다[4, 5]. 지

1) 처리 부품(process component)은 그 기능을 함수 개념으로 파악할 수 있다.

금까지의 연구는 모듈의 시험 또는 시스템 시험에 관한 연구들이 대부분이었다. 통합 검증과 시험에서는 단위 시험이나 시스템 시험 방법을 준용하고 있다. 또한 개발 현장에서는 통합 시 정형화된 방법을 사용하지 못하고 문제 해결을 시스템 시험까지 넘기는 형편이다.

통합 시험에 관한 연구는 일반적인 소프트웨어 모듈 시험을 위해 개발한 여러 가지 기법들, 예를 들면, 문장/가지 범위 시험(statement/branch coverage testing) 방법이나 자료 흐름 분석에 의한 시험 케이스 생성기법 등을 확장하여서 적용하고 있다[7]. 객체지향 개발 방법에서 통합 시험은 주로 Message/Method 경로를 찾아서 시험 케이스를 만드는 것이다[6, 10, 11]. CBD 환경에서 통합시험의 경우 부품 내부를 볼 수 없기 때문에 블랙박스 시험 방법이어야 한다. 따라서 기존 시험 방법들 중 화이트 박스(white-box) 시험 기법들은 사용할 수 없고, 부품 명세와 문서에만 의존하여야 한다. 본 연구에서는 영역 시험(domain testing) 기법을 응용하였다.

국내의 관련 연구로는, UML의 순서도와 협력도를 이용하여 사건 흐름을 구성하는 컴포넌트들 사이의 인터페이스 오류를 추출하는 방법에 관한 연구[16]와, 시나리오를 이용한 객체시스템의 통합시험 방법에 관한 연구[17]가 있다.

3. 처리 부품 및 부품 합성

3.1 처리 부품

[정의 : 처리 부품(Process component)] 처리 부품은 정보의 변환자로서 입력포트(port)와 출력포트를 통해서만 자료를 받아들이거나 내 보낸다. 다시 말하면, 처리 부품은 입력포트로부터 자료를 받아서 정해 진 처리(process)를 거친 후 그 결과를 출력포트로 내보내는 역할을 한다. (이 논문에서는 부품을 강조할 필요가 없을 때에는 단순히 처리기라 부른다.)

처리기의 특징을 살펴보면 다음과 같다.

(1) 처리 부품은 그 기능을 수학적인 함수로 파악할 수 있다는 점에서 객체지향 프로그램에서 나타나는 스택(Stack), 계정(Account)[9] 같은 객체 부품(object component)과 구분된다.

(2) 일반적인 프로그램 언어의 부 프로그램에서는 하나의 가인수(formal parameter)는 한 개의 자료만 받아들이거나 내 보낼 수 있는데 반하여 처리 부품은 하나의 포트를 통해서 연속 자료(data stream)를 입출력할 수 있다.

(3) 처리기에는 포트가 유일한 외부와의 접점이며, 포트를 통하지 않는 부수 효과(side-effect)는 없다.

처리기의 표현 : (그림 2)는 처리기 'SQRT'를 그림으로 나타낸 것이다. 처리기 SQRT는 정수 값을 받아서 그 제곱근을 계산하여 내보내는 것이다.

(그림 2) SQRT 처리기

처리기는 사각형의 상자로 표시하고 그 이름을 상자 안에 기술한다. 들어오는 화살표는 입력 포트를 나타내며 그 위에 포트 명세를 포트 이름과 ' : ' 그리고 자료형의 순서로 기술한다. (예 : x : Integer). 상자에서 나아가는 화살표는 출력 포트를 나타낸다.

처리기의 명세 : 처리기 'C' 의 명세는 인터페이스 명세와 기능 명세로 정의할 수 있다. 인터페이스 명세는 포트 명세들로 구성되며, 포트 명세 는 포트명, 입출력 구분, 자료 형(element type), 포트 형(port type)으로 정의된다. 포트의 형은 하나의 자료만 받아들이는 단순 포트와 연속 자료(data stream)를 받아들일 수 있는 스트림 포트로 구분된다. 스트림 포트의 경우에 포트 이름 다음에 '*'로 표시한다. (예 : x* : in Integer)

기능 명세를 기술하는 방법에 특별한 제한이 없다. 자연어나 수학적 표현(예 : first order predicate)을 사용할 수 있으나 이 논문에서는 선조건(pre-condition)/후조건(post-condition) 형태로 기술하였다. <표 1>은 처리기 'SQRT'의 명세를 보여준다.

<표 1> SQRT의 명세

```
SQRT =
    { Interface
          x : in integer ;
          y : out real ;
      Function : /* 정수를 받아서 그 것의 제곱근 계산한다 */
          pre : x >= 0 ;
          post : y = SQRT(x) ;
          error if x < 0 ;
    }
```

처리기의 구현 : 처리 부품을 실제 프로그램 언어로 구현하는 방법은 크게 두 가지 경우로 나누어서 생각해 볼 수 있다.

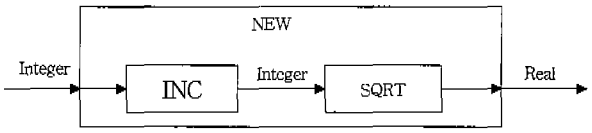
(1) 모든 포트가 단순 포트인 경우 : 대부분의 절차적 언어(예 : Fortran, C, Java 등)에서 부 프로그램으로 쉽게 구현이 가능하다. 출력 포트가 하나 이상인 처리기를 C, Java 등으로 구현할 경우, 부수 효과가 없는 단순 변환은 불가능하고 인수로 자료 값의 포인터를 넘겨서 부 프로그램 내에서 값을 변경시키는 방법을 사용한다.

(2) 스트림 포트가 있는 경우 : CSP(Communicating sequential process)[1], Ada 언어의 Task 형이나, Unix의

프로세스와 파이프를 이용하면 직접 표현이 가능하다. C 언어를 사용할 경우는 스트림 포트의 결과를 외부 파일로 출력하도록 하는 방법이 한 해결책이다.

3.2 처리기의 합성(Composition of process components)

처리 부품의 합성은, 두 개의 부품을 순차적으로 연결하여 새로운 복합 기능을 하는 합성부품을 만드는 것이다. 이러한 통합은 한 부품(공급자라 부름)의 출력포트를 다른 부품(수요자)의 입력포트에 연결함으로써 이루어진다. 이렇게 합성된 부품도 처리기이다. (그림 3)는 두 부품 'INC'와 'SQRT'를 합성한 설계를 보여준다. 처리기 'INC'는 입력포트로 받은 정수를 단순히 '1' 을 증가 시켜서 출력포트로 내보낸다.



(그림 3) 부품의 합성

합성 부품의 명세 : 인터페이스가 잘 정해진 두 개의 부품을 합성할 경우, 합성 부품의 인터페이스 명세는 기계적으로 만들어진다.[2] 한편으로, 합성 부품의 기능 명세는 두 부품 기능의 합성 함수가 된다[18]. 즉,

$$f_{NEW} = f_{INC} \circ f_{SQRT}$$

가 된다. 여기서 기호 "∘"는 함수 합성이다. 실제로는 부품의 기능이 복잡해져서 수학적 함수로 표현할 수 없는 경우가 많다. 이러한 경우에는 각 부품의 비정형 기능 명세로부터 추론해 내어야 한다.

합성된 부품 NEW의 명세는 <표 2>와 같이 표현된다.

〈표 2〉 합성 부품 NEW의 명세

```
New =
    { Interface :
            x : in integer ;
            y : out real ;
      Function :
            Pre : None ;
            Post : y = √(x+1) ;
            Error if x < -1 ;
    }
```

합성의 실제 구현 : 설계서에 따라 두 처리기를 그림과 같이 통합할 경우, 실제 프로그램 언어로 구현하는 방법은 두 가지 경우로 나누어 생각할 수 있다. 즉,

(1) 모든 포트가 단순형일 경우 : 이 경우에는 대부분의 언어에서 존재하는 부 프로그램 호출을 이용하면 간단히 해결된다. 즉, 두개의 부품 INC 와 SQRT를 부 함수로 만들고, 주 프로그램 NEW에서는 단순히 두 부 프로그램을 순서적으로 호출해 주면 된다.[3]

(2) 스트림 포트가 있을 경우 : 앞에서 언급한 것처럼, 각 부품을 Ada 언어의 Task나 운영체제의 프로세스로 만들어서 통합하는 것이 가장 자연스럽다. 만약, 이 통합을 C 언어로 구현한다면, 임시 파일 같은 새로운 자료구조를 활용하여야 할 것이다.

4. 영역 분석

4.1 부품 합성

본 연구에서 상정하는 부품 합성과 검증에 관련된 단계를 정리하면 다음과 같다.

단계 1. 설계서를 작성한다. 설계서는 처리기들과 그 연결 구조를 나타내는 청사진이다.

단계 2. 통합 시스템을 시험하기 위한 시험 케이스를 작성한다

단계 3. 자료형과 기능 측면에서 모순이 없는(compatible) 부품 A와 B를 찾는다. 즉, (1) 연결하려고 하는 두 포트의 형이 합치하고(compatible), (2) 두 기능의 합성결과가 원하는 전체의 기능 명세와 합치한다.

단계 4. 영역 분석 방법으로 통합 시험 케이스를 생성한다.

합성시의 오류 가능성 : 두 부품들을 합성할 때 발생할 수 있는 오류의 가능성과 그 대책은 다음과 같다.

(1) 연결되지 않은 포트가 존재하는 경우 : 쉽게 검증된다.

(2) 두 포트의 자료형이나 포트형이 합치하지 않는(mismatch) 경우 : 쉽게 검증 가능하고, 대부분의 컴파일러에서 검증 해 준다.

(3) 생산자의 실제 치역이 소비자의 실제 정의역과 같지 않은 경우(소비자의 선 조건이나 가정에 맞지 않는 경우 포함)

본 연구에서는 (3)번 경우의 오류에 대하여 집중 분석한다.

처리기의 명세 : 명세에 맞게 구현된 처리기 'C'는

$$C = \langle I_C, F_C, P_C, T_C \rangle$$

로 나타낼 수 있다. 여기서, I_C는 인터페이스 명세, F_C는 기능 명세이고, P_C는 프로그램 모듈이며 T_C는 시험 케이스의 집합이다. 하나의 시험케이스는 입력 값과 그에 해당하는 예상 출력의 쌍으로 표현할 수 있다.

4.2 영역 분석

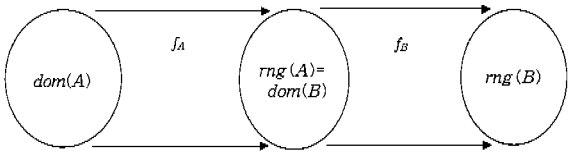
두개의 처리 부품 A와 B를 합성하여 합성 처리기 C를

2) 포트 이름에 대하여 적절한 이름 치환(Renaming) 과정은 필요하다.

3) 약간의 변수 정의 등이 필요하겠지만 거의 기계적으로 합성할 수 있다.

생성할 때, 생산자의 치역과 소비자의 정의역 사이에는 다음과 같은 네 가지 경우가 성립한다.

경우 I : $rng(A)=dom(B)$: 즉 생산자에서 발생 가능한 모든 값이 소비자에서 수용됨. (그림 4)



(그림 4) 경우 I

이때 합성 부품 C의 명세는 다음과 같이 정해진다.

$$dom(C)=dom(A), rng(C)=rng(B),$$
$$f_C=f_A \circ f_B$$

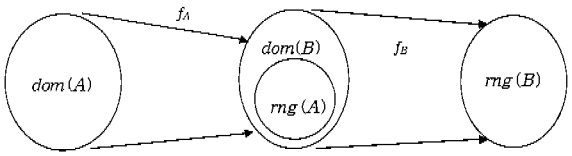
시험 케이스 : 다음의 방법으로 A와 B의 시험 케이스로부터 C의 시험 케이스를 생성할 수 있다.

[방법 1] 모든 $t=\langle t_i, t_0\rangle \in T_A$에 대해서, $\langle t_i, f_B, (t_0)\rangle$는 C의 시험케이스가 된다. 여기서 $f_B(t_0)$는 부품 B를 이용하여 계산할 수 있다.

[방법 2] 모든 $t=\langle t_i, t_0\rangle \in T_B$에 대해서, $\langle f_A^{-1}(t_i), t_0\rangle$는 C의 시험케이스가 된다. 여기서 $f_A^{-1}(t_i)$는 부품 B의 명세로부터 추론이 가능하다.

[분석] 가장 이상적인 통합 경우이나, 실제로 부품의 기능 명세를 검토해 보면, 인터페이스 명세에 기술된 정의역과 치역은(예 : Integer, Real) 실제와는 다른 경우가 대부분이다. 그 원인은, 명세를 정확히 하지 않았거나, 컴퓨터의 단어 길이(word size)와 같은 시스템과 관련된 제한 기인하는 경우 등이 있다. 예를 들면, 부품 SQRT의 치역은 인터페이스 명세에 Real로 되어 있지만 실제 치역은 양의 실수이다.

Case II : $rng(A) \subseteq dom(B)$ (그림 5)



(그림 5) 경우 II

[예] $f_A(x)=x^2+10; f_B(x)=\sqrt{x}$이면 이 경우에 해당한다. 이때 합성 함수 C의 정확한 명세는 다음과 같이 정해진다.

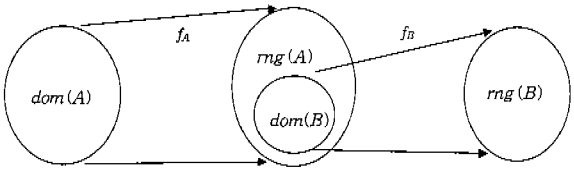
$$dom(C)=dom(A)=\{x \mid x \in int\},$$
$$rng(C)=f_B(rng(A))=\{y \mid y \in real, y \geq \sqrt{10}\},$$
$$f_C=f_A \circ f_B=\sqrt{x^2+10}$$

시험 케이스 : 다음의 방법으로 A와 B의 시험 케이스로부터 C의 시험 케이스를 생성할 수 있다.

(1) 부품 A의 시험 케이스는 위의 [방법 1]에 따라 C의 시험 케이스가 된다

(2) 부품 B의 시험 케이스의 경우도 위의 [방법 2]에 따라 C의 시험 케이스가 될 수 있다. 이 중 입력 값이 $dom(B)-rng(A)$에 속하는 것들은 만약 역상을 만들 수 있다면, C와 A의 오류 영역을 시험할 수 있는 좋은 케이스가 된다.

경우 III : $rng(A) \supseteq dom(B)$ (그림 6)



(그림 6) 경우 III

[예] $f_A(x)=x+10; f_B(x)=\sqrt{x}$ 이면 이 경우에 해당한다. 이때 합성 함수 C의 정확한 명세는 다음과 같이 정해진다.

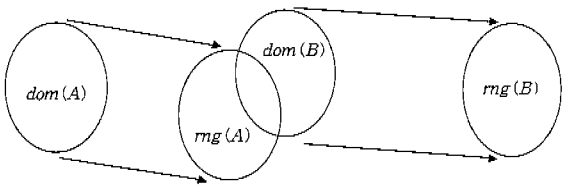
$$dom(C)=f_A^{-1}(dom(B))=\{x \mid x \in int, x \geq -10\},$$
$$rng(C)=rng(B)=\{y \mid y \in real, y \geq 0\},$$
$$f_C=f_A \circ f_B=\sqrt{x+10}$$

시험 케이스 : 다음의 방법으로 A와 B의 시험 케이스로부터 C의 시험 케이스를 생성할 수 있다.

(1) 부품 A의 시험 케이스는 위의 [방법 1]에 따라 C의 시험 케이스가 된다. 이 중에서 예상 출력이 $rng(A)-dom(B)$에 속하는 시험 케이스는 C와 B의 오류 영역을 시험할 수 있는 좋은 케이스가 된다.

(2) 부품 B의 시험 케이스의 경우도 위의 [방법 2]에 따라 C의 시험 케이스가 될 수 있다. 이는 합성 부품 C의 정상 입력 값을 시험하는 케이스가 된다.

경우 IV : (그림 7)과 같이 두 영역이 겹치는 경우(겹치지 않는 경우는 상정할 수 없다. 그런 부품들은 선택하지 않을 것이기 때문에.)



(그림 7) 경우 IV

[예] $f_A(x)=x \bmod (10)$; $f_B(x)=\sqrt{x}$이면 이 경우에 해당한다.

이때 합성 함수 C의 정확한 명세는 다음과 같이 정해진다.

$$dom(C)=f_A^{-1}(rng(A)\cap dom(B))=\{x \mid x\in int\},$$
$$rng(C)=f_B(rng(A)\cap dom(B))=\{y \mid y\in real, 0\le y\le 3.0\},$$
$$f_C=f_A\circ f_B=\sqrt{x \bmod (10)}$$

시험 케이스 : 다음의 방법으로 A와 B의 시험 케이스로부터 C의 시험 케이스를 생성할 수 있다.

(1) 부품 A의 시험 케이스는 위의 [방법 1]에 따라 C의 시험 케이스가 된다. 이 중에서 예상 출력이 $rng(A)-dom(B)$에 속하는 시험 케이스는 C와 B의 오류 영역을 시험할 수 있는 좋은 케이스가 된다. 또한 예상 출력 값이 $rng(A)\cap dom(B)$에 속하는 시험 케이스는 C의 정상 입력을 시험하는 케이스가 된다.

(2) 부품 B의 시험 케이스의 경우도 위의 [방법 2]에 따라 C의 시험 케이스가 될 수 있다. 이 중에서 입력 값이 $dom(B)-rng(A)$에 속하는 시험 케이스는 C와 A의 오류 영역을 시험할 수 있는 좋은 케이스가 된다. 또한 입력 값이 $rng(A)\cap dom(B)$에 속하는 시험 케이스는 C의 정상 입력을 시험하는 케이스가 될 것이다.

4.3 통합 시험자료 T_C의 검증

앞에서 우리는 두 부품을 합성하여 합성 부품을 만들 때, 영역 분석에 따라 네 가지 경우에 대하여, 합성 부품의 명확한 명세를 만드는 방법과, 각 부품들의 시험 케이스로부터 통합 부품의 시험케이스를 생성하는 방법을 보였다. 여기에서는 생성된 시험 케이스가 얼마나 도움이 되는가를 정리하고 설계 시에 만들어진 시험 케이스의 효율성을 검증하는 방법을 제시한다. 여기서 T_C는 설계시에 만들어진 통합부품의 시험 자료이다.

경우 Ⅰ :

(1) T_A, T_B 모두가 통합 시험 자료를 생성하는데 쓰인다.

(2) T_C의 검증 : 시험 케이스의 입력 값이 $dom(A)$ 뿐만 아니라 $dom(A)^C$[4] 에도 있는가를 검증한다.

경우 Ⅱ :

(1) T_A는 모두 통합시험 케이스를 생성하는데 쓰인다.

(2) T_B중에서 $dom(B)-rng(A)$에 속하는 시험 케이스가 $dom(A)^C$에 역상을 가진다면, 즉 그 역상이 A에 관한 우주 집합[5] U_A에 있으면, 이는 C의 오류를 검출할 수 있는 좋은 케이스가 된다.

(3) T_C의 검증 : 시험 자료들이 $rng(A)$와 $dom(B)-rng(A)$를 모두 시험할 수 있는가를 검증한다.

경우 Ⅲ :

(1) T_A는 모두 통합시험 자료를 생성하는데 쓰이며, 그 중 예상 출력이 $rng(A)-dom(B)$에 속하는 시험케이스는 부품 B의 영역 오류를 검출하는데 도움이 된다.

(2) T_B중에서 입력 값이 $dom(B)$에 속하는 케이스에서 생성된 시험 케이스는 C의 정상 동작을 시험하는 케이스가 되고, $dom(B)^C$로부터 생성된 시험 케이스는 C의 영역 오류 검출에 도움이 된다.

(3) T_C의 검증 : 시험 케이스들이 $dom(B)$와 $dom(B)^C$를 모두 지나가도록 되었는지 검증한다.

경우 Ⅳ :

(1) 이 경우는 경우 Ⅱ와 경우 Ⅲ의 혼합 형태로서 같은 논리를 적용할 수 있다.

(2) T_C의 검증 : 시험 케이스가 (그림 7)의 가운데 표시된 세 영역 ($rng(A)-dom(B)$, $dom(B)-rng(A)$, $rng(A)\cap dom(B)$)를 모두 시험하는 가를 검증한다.

결론 및 관찰 : 이상에서 합성에 참여하는 두 부품의 영역을 분석해 봄으로써, 합성 부품의 명세를 명확히 하는 방법과, 개별 부품의 시험 케이스로부터 합성 부품의 시험케이스를 만드는 방법을 제시하였으며, 설계 시에 만들어진 시험케이스 (T_C)를 영역 시험의 관점에서 유효성을 검증하는 방법을 제시하였다. 이 과정에서 우리는 다음과 같은 중요한 교훈을 얻었다.

(1) 인터페이스에 명시된 영역(정의역, 치역)은 명세서 편의상 사용하는 영역으로 정확한 영역이 될 수 없다. 따라서 부품의 명세와 선 조건을 분석하여 통합 부품의 영역을 명확히 해야 한다.

(2) 생산자의 시험 자료로부터 통합 부품의 시험자료를 생성하는 것은 쉽다.[6] 그러나 소비자의 시험자료로부터 합성 부품의 시험 자료를 얻는 방법은 역상을 추론하여야 하므로 어렵고 생산자의 우주 집합에도 존재하지 않는 경우도 있다.

(3) 통합 부품의 시험 자료를 생성하는 데에는 생산자의 시험 케이스가 소비자의 시험 케이스 보다 더 효용성이 높다.

5. 실 험

앞 장에서 보인 바와 같이, 수(Number)만을 다루는 부품의 합성은 이 방법을 쉽게 적용할 수 있다. 본 연구에서는

4) A^C는 집합 A의 여집합.

5) 부품 A에 입력 가능한 모든 요소들의 집합.

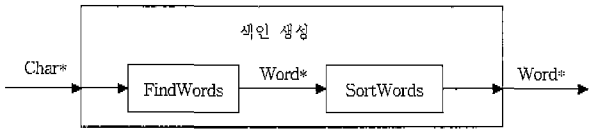
6) 예상 출력을 소비자 부품을 실행시켜서 계산해 내거나 명세에 따라 쉽게 추론 할 수 있으므로

다른 분야 적용가능성을 보이기 위하여 연속 자료(문자 스트림)를 다루는 분야에서 다음과 같은 실험을 하였다.

문제 : 보고서를 작성한 후 마지막에 첨가할 인덱스(Index)를 작성하려고 한다. 따라서 원하는 시스템은 텍스트를 읽어서 보고서에 나오는 모든 단어들을 알파벳 순서로 나열된 리스트를 보고 싶다.

실험 : 다음과 같은 순서로 실험을 진행하였다.

(1) 시스템을 개발하기 위하여 다음과 같은 설계서를 작성하였다. (그림 8)



(그림 8) 색인 생성 설계서

(2) 설계서 작성 후, 두 부품의 명세를 대강 기술하고(부록 참조)

(3) 두 사람 P와 Q에게 두 부품의 명세를 주고, 이에 맞는 부품을 C 언어로 개발하게 하였다. 또한 각각 다섯 개 이상의 시험 케이스를 만들어서 시험하고, 시험 자료도 제출하도록 하였다. 이때 부품의 사용 용도는 설명하지 않았다.

(4) 그 동안 우리는 통합 시험 자료를 만들었다. 인터넷에서 영어로 된 기사 두 개만을 구해 두었다.

(5) 두 사람에게서 가져온 부품 중에서, 임으로 부품 A는 P의 것을, 부품 B는 Q의 것을 선택하여 통합 및 시험을 진행하였다.

분석 : 두 부품 A와 B의 명세와 헤더파일을 검토한 결과 다음과 같은 사항을 발견했다. 즉,

(1) 두 부품 모두 텍스트 파일을 입력과 출력으로 삼았다.

(2) 부품 A는 원하는 영어 단어(뒤에 숫자가 나오는 경우 포함)을 인식해 내었고, 부품 B는 영 단어뿐만 아니라 모든 문자열을 유효한 입력으로 받아들여서 기능을 수행했다.

(3) 부품 A는 출력단어의 최대 길이를 100문자로 하였고, 부품 B는 최대 단어 길이를 30문자로 제한하여 개발하였다.

이는 두 부품이 두개의 다른 차원에서 모순을 보여주었다. 즉, 단어 길이에서는 위의 영역분석의 [경우 III]에 해당되고, 유효한 단어 집합에서는 [경우 II]에 해당하는 결과가 나온것이다.

통합 : 먼저 두 함수를 통합하기 위하여 단어 길이를 통일해야 했다. 이것은 간단히 두 부품의 헤더파일에서 단어 길이를 모두 30으로 고쳤다. 유효한 단어 집합에 관한 불합치에 대해서는, 두 부품의 관계가 [경우 III]에 해당 하므로 문제가 없었다.

통합 부품의 시험 자료 생성 : 부품 A의 시험 자료는 모두 사용하기로 했다. 한편으로, 부품 B의 시험 자료는 다음과 같이 함수 f_A에 대한 역상을 만들어서 사용하였다. 즉, 부품 B의 모든 시험 자료의 입력 값 문자열에 대해서 그 사이에

(1) 빈칸들을 삽입하여 텍스트 문서를 만들었다.

(2) 분리 문자[7]들의 문자열을 임으로 생성하여 삽입하였다.

부품 A의 시험 자료로부터 합성 부품의 예상 출력을 만들어 내는 일은 어려운 일은 아니었으나, 부품 B의 시험 자료로부터 역상을 만들어 내는 일은 까다로웠다. 최종적으로 우리가 만든 두 개의 시험자료를 포함 총 17개(T_A로부터 다섯 개, T_B로부터 10개)의 시험 자료를 만들어서 사용하였다.

6. 결 론

이 논문에서는 소프트웨어 구조를 자료 흐름 모델(Data-flow systems)로 설계하고 부품을 통합할 경우에 통합을 효과적으로 검증할 수 있는 방안을 제시하였다. 합성에 참여하는 두 부품의 영역관계를 분석해 봄으로써, 합성 부품의 명세를 명확히 하는 방법과, 개별 부품의 시험 케이스로부터 합성 시스템의 시험케이스를 만드는 방안을 제시하였다. 이와 같이 생성한 시험 케이스는 이미 만들어진 통합 시스템의 시험 케이스의 효과를 강화 시켜 준다. 또한 영역분석 방법은 이미 만들어진 통합 시스템의 시험 케이스의 효율성을 검증하는 것도 가능하게 한다. 우리는 실험을 통하여 숫자를 영역으로 하는 부품뿐만 아니라, 다른 응용 분야에서도 적용이 가능하다는 것을 보였다.

이 방법은 자료흐름 시스템 구조 뿐만 아니라, 심각한 부수 효과가 없는 '주 프로그램-부 프로그램'(Main program and subprogram) 구조와, 실시간 제한 측면을 제외하는 경우 '통신 프로세스'(Communicating Processes) 구조에도 적용할 수 있다. 여기에 제시된 방법에서 핵심 결과인 각 부품의 시험 케이스를 이용하여 합성 시스템의 시험 케이스를 만들어 내는 방법은 현실에서 큰 효과를 보일 것으로 예상된다.

본 논문에서는 입력포트나 출력포트가 각각 하나씩 존재할 경우만을 보였다. 입/출력포트 수가 두개 이상일 경우에 대해 일반화하는 것도 가능할 것이다. 또한 포트의 연결을 공유 파일이나 공유 자료 구조를 사용하는 경우에도 여기

7) 빈칸, 줄 바꿈 문자, 수평 탭 문자, 영문자나 숫자가 아닌 ASCII 문자들

에 제안한 방법은 특별한 수정 없이 그대로 적용될 수 있다. 이는 매우 현실적으로 고려할 수 있지만, 미리 그 공유 자료 구조를 정하고 설계서를 작성하여야 하며, 부품의 구현도 그 방법을 따라야 한다는 제약점이 있다.

향후 계획은 우선, 이 논문에 제시된 기법들이 자료 흐름 구조로 설계되는 모든 시스템에 적용 가능함을 입증하기 위하여, 다른 응용 분야에 대해서도 개발 실험을 수행하고 발생 가능한 문제점을 분석할 예정이다. 또한 영역 분석 방법을 확장하여 객체지향 구조의 시스템 개발에도 적용할 수 있도록 할 예정이다.

참 고 문 헌

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

[2] M. Shaw and D. Garlan, 'Software Architecture : Perspectives on an emerging discipline,' Prentice Hall, 1996.

[3] Mary Shaw, Comparing Architectural Design Style, IEEE Software, pp.27-41, Nov. 1996.

[4] B. Beizer, 'Software Testing Techniques,' 2nd Ed., Van Nostrand Reinhold, 1990.

[5] James A. Whittaker, "What Is Software Testing? And Why Is It So Hard?," IEEE Software, pp.70-79, January, 2000.

[6] Y. G. Kim, et al, Test cases generation from UML state diagrams, IEE Proc. Software, Vol.146, No.4, pp.187-192, August, 1999.

[7] M. J. Harrold and M. L. Soffa, "Selecting and Using Data for Integration Testing," IEEE Software, pp.58-65, March, 1991.

[8] Elaine J. Weyuker, "The Component-Based Software : A Cautionary Tale," IEEE Software, pp.54-59, Sept./Oct. 1998.

[9] Robert V. Binder, Testing Objects : State-Based Testing, Object Magazine, July/August and Sept./Oct. 1995.

[10] Paul C. Jorgensen and Carl Erickson, Object-Oriented Integration Testing, CACM, pp.30-38, September, 1994.

[11] Gail C. Murphy, et al, Experiences with Cluster and Class Testing, CACM, pp.39-47, Sept. 1994.

[12] Alan W. Brown and Kurt C. Wallnau, "The Current Status of CBSE," IEEE Software, pp.37-46, Sept./Oct. 1998.

[13] Michael Sparling, Lessons Learned Through Six Years of Component-Based Development, CACM, pp.47-53, October, 2000.

[14] D. Garlan, et. al, "Architectural Mismatch : Why Reuse is So Hard," IEEE Software, pp.17-26, Nov. 1995.

[15] Amy M. Zaremski and Jeannette M. Wing, Specification Matching of Software Components, ACM TSEM 6(4), pp.333-369, October, 1997.

[16] 윤희진 등, "UML 기반 콤포넌트 통합 테스팅", 정보과학회 논문지(B), 제26권 제9호, pp.1105-1113. 1999.

[17] 김은주, 최은만, "시나리오를 이용한 객체 지향 시스템의 통합 테스트", 한국정보처리학회논문지, 제5권 제9호, pp.2312-2322, 1998.

[18] 김대수, 장재건, 전산 수학, 생능출판사, 2000.

부록. 실험에 사용된 부품의 명세

A.1 부품 'FindWords'의 명세

```
FindWords =
{Interface
        ch* : in character ;
        Word* : out string ;
Function
 Pre : ch 는 ASCII characters ;
 Post :
 /* (1) 출력(단어 스트림)은 입력(문자 스트림)을 순서대로 읽어서 유효한 단어를 갈 이루는 string을 순서대로 찾아낸 것이다. (2) 유효한 단어는 Alphabet 문자로 시작하고 문자 또는 숫자로 구성된 연속하는 문자열을 말한다. (3) 단어 중간에 '-' 문자를 하나 포함할 수 있다. (4) Break character들은 무시되며, 단어 형성 중에 나타나면 그 단어는 끝난다. */
}
```

A.2 부품 'SortWords'의 명세

```
SortWords =
 {Interface
      Word* : in string ;
      Index* : out string ;
 Function
      Pre :  /*   Word는 유효한 영어 단어. */
      Post : /*   입력 단어들을 Lexical sorting 한 단어들의 스트림. */
}
```


주 복 규

e-mail : bkjoo@wow.hongik.ac.kr

1977년 서울대학교 계산통계학과 졸업

1980년 한국과학원 전산학과 졸업(이학석사)

1990년 University of Maryland 졸업 (공학박사)

1990년~1998년 삼성종합기술원, 삼성전자 중앙연구소 수석연구원

1998년~2000년 동양시스템즈 연구소장

2000년~2001년 한국과학기술원 전산학과 초빙교수

2001년~현재 홍익대학교 전자전기컴퓨터공학부 교수

관심분야 : Software Reuse, Software Testing, Network Security, Intelligent Systems

김 영 철

e-mail : bob@wow.hongik.ac.kr

1985년 홍익대학교 전산학과 졸업

1987년 광운대학교 전산학과 졸업(이학석사)

2000년 Illinois Institute of Technology 졸업(이학박사)

2000년~2001년 LG 산전 중앙연구소 책임연구원

2001년~현재 홍익대학교 전자전기컴퓨터공학부 교수

관심분야 : Use Case 방법론 및 툴 개발, Design driven Testing, Maturity models, 데이터 모델링, Information Architecture, Mobile agent