韓國 OA學會 論文誌
第6卷 第4號, 2001. 12.

2001-6-4-1-3

소프트웨어 재사용을 위한 소프트웨어 칩 표현식에 관한 연구

김 홍 진*

A Study on the Software-chip Expression for Software Reuse

Hong-Jin Kim*

요 약

소프트웨어 수요와 공급의 불균형으로 소프트웨어 병목현상이 나타나고 있으며, 이는 기존의 소프트웨어를 작성하는 프로그래머의 능력을 향상시키지 못함에 기인한다. 그러므로 소프트웨어 생산성 향상을 위해서는 새로운 소프트웨어 작성방법이 필요하다.

본 논문은 소프트웨어 재사용을 위해 표준화시켜서 작성한 프로그램 모듈들을 소프트웨어 칩으로 사용하기 위한 표현식을 제안한다.

이 표현식은 각 소프트웨어 칩의 이름과 입력, 출력, 반복의 4개 요소로 구성되어 직렬 및 병렬, 반복과 복합, 혼합, 다중 형태의 결합과 분리되는 관계를 간단히 표현할 수 있고, 데이터의 흐름을 명확히 파악할 수 있으며 소프트웨어 재사용을 쉽게 할 수 있다.

Abstract

The problem of software bottle-neck may be arised from unbalance of demands and supply of software. This is caused from the fact that the capability of programmer could not be improved in software development. Therefor, the new method of software development should aim at improving the productivity of software.

This paper presents the expressions to be the standardized software program modules by means of the software chip.

The expressions are consist of name, input, output, and iteration of each software chip. And they simple express a combination and separation in sequence, parallel, iteration, composition, mixing, and variety form. Therefore they can easily software reuse as a result of analyzing the flow of data clearly.

* 경원전문대학 컴퓨터정보과 교수

Ⅰ. 서론

컴퓨터의 이용범위가 확대되면서 소프트웨어의 수요는 연평균 21~23% 정도씩 증가하고, 전문적인 프로그래머의 수도 연평균 13%씩 증가하여 왔다.[11] 그러나 소프트웨어의 생산성은 겨우 3~8% 정도씩 증가하여 왔다.[13, 15] 이러한 수요와 공급의 불균형 현상을 해결하기 위해서는 소프트웨어의 생산성을 향상시킬 수 있는 새로운 소프트웨어 작성 방법을 개발해야 한다.

소프트웨어 재사용은 소프트웨어의 품질과 생산성을 개선하여 소프트웨어의 개발 기간을 단축하며, 개발과 유지보수 비용을 절감하여 소프트웨어의 개발 환경을 개선하는데 기여한다.[1]

소프트웨어 재사용의 개념은 서브루틴 라이브러리가 제안된 이후 프로그래밍에서 상속되었고,[14] 1967년 McIlroy가 소프트웨어 부품(component)의 등록을 계획하였으며, 이의 응용이 Lanergan과 Poynton에 의해 제한된 영역안에서 수행되었다.[6]

1983년 New Port에서 개최된 "Workshop on reusability in programming"에서는 이 방법이 소프트웨어의 생산성과 질을 향상시키기 위한 해결책으로 지적되었고,[12] 소프트웨어 재사용성(reusability)의 타당성은 여러 연구에서 검증되었다.[6, 11, 14]

소프트웨어의 재사용 방법중에서 특정 기능을 수행하는 프로그램 모듈을 하드웨어의 부품과 같이 부품(component)화 시켜서 사용하는 방법은 수학적 함수나 운영 체제 및 자동화 기계용 프로그래밍과 같이 변화가 적은 분야에서 효과적으로 이용할 수 있으며, Woodfield는 이 방법에 의하여 생산성을 35 - 80% 까지 향상시킬 수 있다고 하였다.[8]

소프트웨어를 부품화 시키는 방법을 위하여 Lamar Ledbetter와 Brad Cox는 소프트웨어 IC를 제안하였으며[3], Michael F. Korns는 소프트웨어 버스를 제안하였다.[5]

소프트웨어를 부품화하여 재사용하는 것은 이들 부품들이 어떻게 결합되었는지의 관계를 간편히 표현할 수 있어야 하며, 각 부품들을 표현되어 있는 관계에 따라 효과적으로 결합하는 도구(tools)의 개발이 필요하다.

Browne는 단위 소프트웨어를 하드웨어의 부품과 같은 개념으로 생각한 소프트웨어 칩(software chip)으로 다루어 CODE 프로그래밍 환경에서 ROPE에 적용하였다.[9] Browne이 사용한 다양한 소프트웨어 칩의 결합 관계를 표현한 기법은 기본적 결합관계를 명확하게 나타낼 수는 있지만 도식적 결합관계 표현방법이어서 프로그램 작성시에 직접적인 표현 관계를 표현하기 어려운 문제점을 가지고 있다.

본 논문에서는 단위 기능을 수행하는 프로그램 모듈들이 라이브러리에 저장된 환경에서 이들 각각을 소프트웨어 칩으로 취급하고 이들의 결합 관계를 간편히 표현하는 방법을 제안한다. 또한, Browne이 사용한 소프트웨어 칩의 결합 개념을 확장시키고 새로운 소프트웨어 작성시에 각 소프트웨어 칩들을 논리적으로 결합할 수 있는 표현 방법을 제안하고 Diaz와 Lenz 및 Browne이 제안한 표현 방법을 비교하여 분석하였다.

Ⅱ. 소프트웨어 부품 재사용

2.1 소프트웨어 부품

소프트웨어 부품(component)은 소프트웨어를 다른 여러 시스템을 구축하는데 일부분이 되는 단위로 작성하여 라이브러리에 저장시켜 놓고 새로운 소프트웨어를 작성할 때 필요한 것을 하드웨어의 조립에서 부품을 결합하는 것과 같이 이들을 결합하여 사용하도록 하는 것이다.

소프트웨어 부품 재사용 방법은 1968년 McIlroy가 소프트웨어 부품 설비(software component facility) 계획을 발표하면서 본격적으로 연구되었다.[7, 11, 15] 소프트웨어를 부품화하여 사용하는 것은 재사용 율을 높여주어 소프트웨어 생산성을 향상시킬 수 있다.

2.2 소프트웨어 칩

소프트웨어 칩은 소프트웨어를 작성할 때 재사용을 위하여 라이브러리에 저장시켜 놓은 소프트웨어 부품을 집적회로의 칩과 같이 취급하는 방법이며, Lamar Ledbetter

와 Brad Cox의 소프트웨어 IC에서 제안되었다.[3]

소프트웨어 칩은 하드웨어에서 유일한 기능을 수행하는 칩들의 많은 구성요소가 사용자의 요구에 의하여 결합되어 필요한 기능을 수행하도록 하며, 요구자는 칩 내부의 처리 방식이나 데이터에 대해 알 필요가 없는 것과 같은 개념으로 사용하는 것이다.[2, 10]

소프트웨어 칩은 하드웨어에서 처럼 표준화된 부품의 기능이 잘 정의되고 식별이 쉽도록 하여 새로운 시스템을 위하여 통합되어 질 수 있도록 하고, 소프트웨어 재사용 방법을 보다 쉽게 한다.

2.3 소프트웨어 칩의 결합

소프트웨어 칩의 결합은 재사용을 위하여 소프트웨어 칩들이 마련되어 라이브러리에 저장된 환경에서만 가능하다.

(정의 1) 소프트웨어 칩의 결합은 각각 다른 기능을 수행하는 2개 이상의 소프트웨어 칩을 연결하는 것으로 정의한다.

소프트웨어 칩을 사용하여 새로운 소프트웨어를 작성할 때는 필요한 기능을 수행하는 소프트웨어 칩을 요구하는 기능이 수행될 수 있도록 결합하는 것이다.

2.4 소프트웨어 부품의 표현

소프트웨어 재사용을 어렵게 하는 이유 중의 하나는 재사용되는 부품과 그들의 관계를 명세하는 도구의 부족과[1, 14] 표준화된 문법이 없다는 점이다.[3] 이러한 문제를 해결하기 위해서는 소프트웨어의 부품과 그들의 결합 관계를 명확히 나타낼 수 있는 일반적인 표현 방법이 필요하다. Diaz는 소프트웨어의 재사용되는 부분을 표현하는 기술자(descriptor)를 기능부와 환경부의 두 부분으로 구분하여, 기능은 〈기능, 대상, 매체〉로 나타내고, 환경부는 〈시스템 형태, 기능 장소, 응용되는 곳〉으로 표현하는 방법을 제안하였다.[7]

Lenz는 재사용되는 부분을 〈기능, 환경〉의 두 가지 요소를 기본으로 하고, 이들 요소는 BBLX언어를 사용하여 다시 세부 사항을 명세하는 방법으로 표현하였다.[4]

Browne은 소프트웨어 칩의 개념을 사용하여 각 소프트웨어 칩들이 결합되는 관계를 직렬은 ||, 병렬은 □ 기호로 나타내고, 분리는 현재의 환경과 R 기호로 표현하는 방법을 사용하였다.[9]

Diaz와 Lenz의 표현 방법은 일반화가 어렵고 복잡하며, 하나의 프로시저(procedure)를 선언해야 하는 불편함이 있다.

Browne의 표현 방법은 소프트웨어 칩들에 대한 결합과 분리의 관계를 구체적이고 간략하게 표현할 수 있으나 데이터의 흐름을 나타낼 수 없고, 혼합적 결합이나 반복 구조를 나타낼 수 없으므로 논리적인 관계를 표현하기가 어렵다.

Ⅲ. 소프트웨어 칩의 표현

소프트웨어 칩들을 요구하는 기능이 수행되도록 결합하기 위해서는 결합과 분리에 대한 논리적인 관계를 명확하게 표현할 수 있어야 한다.

(정의 3) 소프트웨어 칩은 재사용을 위하여 작성한 하나 이상의 데이터를 입력과 출력하는 기억 장소를 가지고 있는 독립적인 소프트웨어 모듈로 정의한다.

소프트웨어 칩은 그 내부의 처리 과정은 몰라도 되므로 어느 데이터를 처리하는 지와 결과를 어디로 출력하는 지를 명확하게 표현하여야 한다. 단, 자신의 출력 결과를 다시 입력하여 처리하는 것을 필요한 회수만큼 반복하는 과정도 표현할 수 있어야 한다.

(정의 4) 기본 소프트웨어 칩은 칩 이름, 입력, 출력, 반복의 4개 요소로 구성하고 표현식 (1)과 같이 표현한다.

{ 칩 이름 [입력] [출력] 반복 } ----------(1)

이와 같은 칩의 구조는 그림 1과 같으며, 입력은 그 칩이 처리할 데이터 값을 전달받는 곳이고, 출력은 처리 결과를 기억시키는 장소이다.

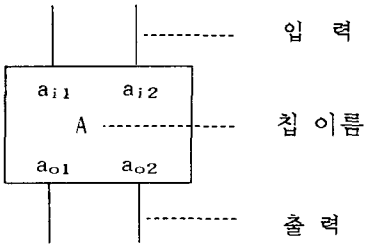


그림 1. 기본 소프트웨어 칩
fog. 1 basic software chip

(정의 5) 각 소프트웨어 칩은 단위 기능을 1번 이상

수행시키기 위하여 결합하는 것이므로 반복 요소가 1인 경우에는 생략할 수 있도록 한다.

(정리 1) 기본 소프트웨어 칩이 결합되어 이루어진 독립된 하나의 단위는 기본 소프트웨어 칩과 같이 사용할 수 있다.

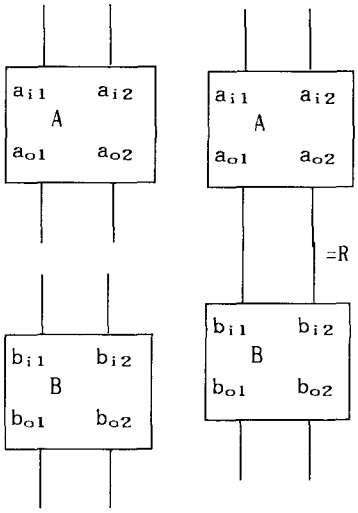
(증명) 기본 소프트웨어 칩이 결합되어 독립된 하나의 단위가 되었을 때 그 자체 내에서의 소프트웨어 칩들 간의 관계에 상관없이 최초 입력 데이터에 따라 출력되는 결과는 일정하다. 그러므로 내부의 결합관계는 변수에 의한 데이터 전달과 같고, 전체는 최초의 입력과 최종결과로 나타나는 하나의 소프트웨어 모듈과 같으므로 기본 소프트웨어 칩과 동일하다.■

3.2 소프트웨어 칩 결합의 표현

소프트웨어 재사용에 있어서 소프트웨어 칩의 결합은 가장 중요한 부분중의 하나이며, 직렬과 병렬, 다중, 반복 구조와 이들의 조합에 따른 다양한 형태가 성립한다.

3.2.1 직렬 결합의 표현

어떤 소프트웨어 칩에서 출력한 결과를 다른 소프트웨어 칩에서 입력하는 관계가 차례대로 수행되어 선형 구조를 이루는 것을 직렬결합이라 한다.



a) 기본 소프트웨어 칩 b) 결합된 소프트웨어 칩

그림 2. 직렬 결합
fig. 2 serial composition

그림 2는 A 와 B 두개의 소프트웨어 칩을 직렬로 결합한 새로운 소프트웨어 칩 R 을 보여 준다.

(정의 6) 소프트웨어 칩의 결합에서 2개 이상의 입력과 출력이 있을 때 이들을 쉼표로 구분하고 같은 순번끼리 대응되어 데이터 값을 전달하는 것으로 정의한다.

(정의 7) 소프트웨어 칩이 계층적으로 결합될 때 동일 계층 내에서는 "," 기호 , 계층간은 ":" 기호로 각 소프트웨어 칩을 구분하여 표현하는 것으로 정의한다.

그림 2와 같은 2개의 소프트웨어 칩을 직렬로 결합하여 새로운 칩 R을 형성하는 것은 정의4 ~ 정의7 에 의하여 다음과 같이 표현한다.

R = { A [ai1, ai2] [ao1, ao2]: B [bi1, bi2] [bo1, bo2] } ----- (2)

(정리 2) 소프트웨어 칩이 직렬구조로 결합될 때 앞 단계의 출력에 대한 표현은 생략하고, 다음 단계에 대응되는 입력에 그것을 표현한다.

(증명) 정의 3에 의하여 각 소프트웨어 칩은 독립된 입출력 장소를 가지고 있으므로 앞 단계의 출력을 다음단계의 입력에 표현하는 것은 변수끼리의 데이터를 전달하는 것과 같다. 그러므로 앞 단계의 출력을 대응되는 다음 단계에서 직접 입력으로 표현하여도 같은 결과가 된다.■

표현식 (2)는 정리 2에 의하여 표현식 (3)과 같이 표현할 수 있다.

R = { A [ai1, ai2]: B [ao1, ao2] [bo1, bo2] } ----------(3)

소프트웨어 칩 A와 B가 직렬로 결합된 새로운 소프트웨어 칩 R은 입력이 [ai1, ai2]] 이고, 출력이 [bo1, bo2]인 기본 칩과 같으므로 정리 1에 의하여 표현식 (3)은 표현식 (4)와 같이 표현할 수 있다.

{ R [ai1, ai2]] [bo1, bo2] } ----------(4)

3.2.2 병렬 결합의 표현

2개 이상의 소프트웨어 칩이 동일한 시간에 각각의 데이터를 처리하여 그 결과를 직렬적인 다음 단계로 전달하는 구조에서 동일 계층에 있는 소프트웨어 칩의 집합을 병렬 결합이라고 한다.

그림 3은 두개의 소프트웨어 칩 A와 B가 병렬로 결합되어 새로운 소프트웨어 칩 R을 만드는 것을 보여 준다.

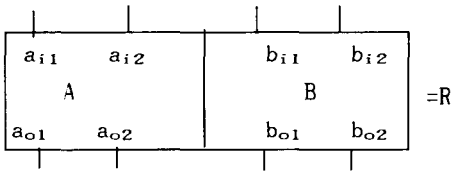


그림 3. 병렬 결합
fig. 3 parallel composition

그림 3과 같이 2개 이상의 소프트웨어 칩이 병렬로 결합된 것은 정의4~ 정의7에 의하여 표현식 (5)와 같이 표현한다.

R = { A [ai1, ai2] [ao1, ao2], B [bi1, bi2] [bo1, bo2] } ——(5)

소프트웨어 칩 A와 B가 병렬로 결합된 새로운 소프트웨어 칩 R의 입력은[ai1, ai2] 와[bi1, bi2] 이며, 출력은 [ao1, ao2]와 [bo1, bo2]이므로 정리 1에 의하여 표현식 (5)는 표현식 (6)와 표현할 수 있다.

{ R [ai1, ai2, bi1, bi2] [ao1, ao2, bo1, bo2] } ————(6)

(증명) 새로운 소프트웨어 칩 R의 입력은[ai1, ai2] 와[bi1, bi2] 이며, 출력결과는 [ao1, ao2]와 [bo1, bo2]이므로 정리 1에 의하여 기본 소프트웨어 칩 R은 표현식 (6)과 같이 표현할 수 있다.

3.2.3 다중 결합의 표현

하나의 소프트웨어 칩의 출력 결과가 두개 이상의 소프트웨어 칩에 입력되는 구조로 결합된 것을 다중 결합이라고 한다.

그림 4는 소프트웨어 칩 A의 출력 결과가 소프트웨어 칩 B, C, D, E로 입력되는 다중 결합을 보여준다.

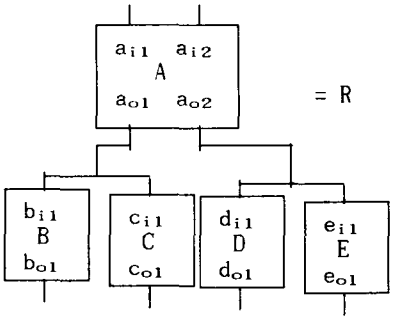


그림 4. 다중 결합
fig. 4 multiple composition

그림 4와 같은 다중 결합은 정의4 ~ 정의7 및 정리 2에 의하여 표현식 (7)과 같이 표현한다.

R = { A [ai1,ai2]: B [ao1][bo1,bo2], C [ao1][co1], D [ao2][do1], E [ao2][eo1, eo1] } ————(7)

소프트웨어 칩 A가 B, C, D, E와 다중 결합된 새로운 소프트웨어 칩 R의 입력은 [ai1,ai2]이고, 출력은 [bo1,bo2], [co1], [do1], [eo1,eo2]인 기본 소프트웨어 칩과 같으므로 정리 1에 의하여 표현식 (7)은 표현식 (8)과 같이 표현할 수 있다.

{ B [ai1,ai2] [bo1,bo2,co1,do1,eo1,eo2] ————(8)

3.2.4 복합 결합의 표현

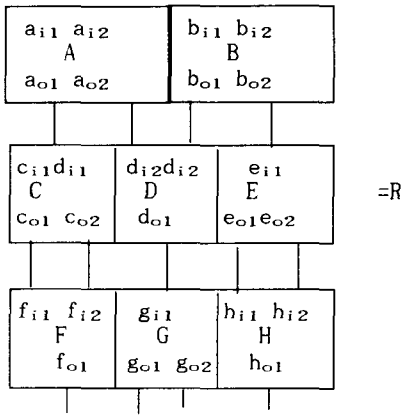


그림 5. 복합 결합
fig. 5 mixing composition

병렬적으로 결합된 소프트웨어 칩이 병렬적으로 결합

된 다른 소프트웨어 칩에 계층적인 구조로 결합된 것을 복합 결합이라 한다.

그림 5는 병렬적으로 결합된 3개의 단위 소프트웨어 칩이 계층적으로 결합된 복합 결합의 구조를 보여준다

그림 5와 같이 복합적으로 결합된 구조는 정의4 ~ 정의7 및 정리 2에 의하여 표현식 (9)와 같이 표현한다.

R = { A {ai1, ai2}, B {bi1, bi2}; C {ao1}, D {ao2, bo1}, E {bo2};
F {co1, co2} {fo1}, G {do1} {go1, go2}, H {eo1, eo2} {ho1} } ————————(9)

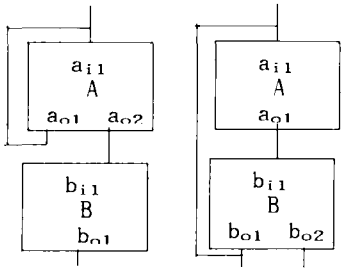
복합적으로 결합된 새로운 소프트웨어 칩 R은 입력이 {ai1, ai2}이고, 출력이 {fo1}, {go1, go2}, {ho1}인 기본 소프트웨어 칩과 같으므로 정리 1에 의하여 표현식 (9)는 표현식 (10)과 같이 표현할 수 있다.

{ R {ai1, ai2} {fo1, go1, go2, ho1} } ————————(10)

3.2.4 반복구조의 표현

어떤 소프트웨어 칩이 자신의 출력을 입력하여 처리하는 것을 단독 또는 다른 소프트웨어 칩들과 같은 시간에 동일 계층이 함께 수행하는 것을 반복구조라 한다. 단일 소프트웨어 칩에 대한 반복은 정의 4에 의하여 각 소프트웨어 칩에 표현하면 되지만 직렬 결합된 전체의 반복이나 병렬 결합에서 하나의 계층 전체의 반복은 괄호로 묶어서 표현한다.

그림 6은 직렬 결합에서 단일 소프트웨어 칩이 먼저 반복 수행한 후 다음 계층으로 진행되는 것과 결합된 전체가 반복되는 것을 보여준다.



a) 단일 반복 b) 직렬 전체의 반복
그림 6. 직렬적 반복 구조
fig. 6 iteration composition

(정의 8) 반복 구조의 표현에서 다음 단계의 입력으로 사용되는 출력은 각 소프트웨어 칩 출력의 가장 앞부분에 표현한다.

그림 6의 a)는 먼저 소프트웨어 칩 A가 주어진 횟수만큼 반복된 후 B로 진행되는 것이며, 이것은 앞의 정의에 의하여 표현식 (11)과 같이 표현한다.

R = { A{ai1} {ao1} n; B {ao2} {bo1} }————(11)

그림 6의 b)는 소프트웨어 칩 A와 B가 직렬로 결합된 상태에서 n회 반복되는 구조이며, 이것은 앞의 정의에 의하여 표현식 (12)와 같이 표현한다.

R = { (A {ai1}; B {ao1} {bo1,bo2}) n } ————(12)

병렬적인 결합에서 특정 소프트웨어 칩이 단독으로 반복되는 것은 정의 4에 의하여 그 소프트웨어의 반복요소에 표현하면 되지만, 하나의 계층 전체의 반복은 각각의 소프트웨어 칩이 단독으로 반복되는 수가 동일한 경우에 한하며 , 괄호로 계층전체를 묶어서 표현한다.

그림 7은 소프트웨어 칩 A와 B 가 병렬 결합된 상태에서 전체가 반복되는 것을 나타낸 것이며, 이것은 표현식 (13)과 같이 표현한다.

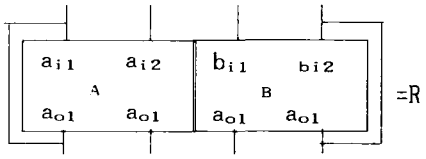


그림 7. 병렬적 반복 결합
fig. 7 iteration of parallel composition

R = { (A {ai1}, {ao1,ao2}, B {bi1} {bo2,bo1}) n } ————(13)

3.2.5 혼합적 결합의 표현

직렬과 병렬 및 반복구조가 함께 조합되고, 특정 소프트웨어 칩은 다음 계층을 건너뛰어서 결합되거나, 복합적 결합에서 서로 엇갈리게 데이터가 전달되는 등의 순수한 직렬과 병렬, 복합, 반복구조 및 다중 구조가 아닌 모든 구조의 결합을 혼합적 결합이라고 한다.

그림 8은 계층을 건너뛰는 형태의 혼합적 결합 관계를 보여준다.

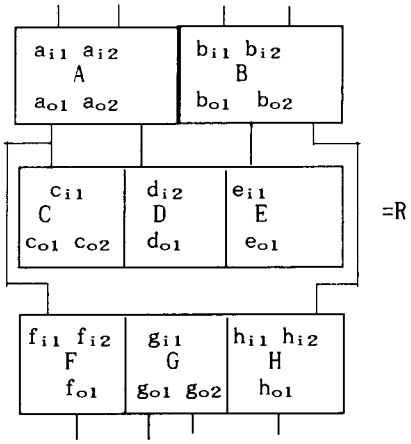


그림 8. 계층을 건너 뛴 혼합적 결합
FIG. 8 jumping in mixing composition

그림 8과 같은 혼합적 구조는 정의4 ~ 정의7 및 정리 2에 의하여 표현식 (14)와 같이 표현한다.

R = { A [ai1, ai2], B [bi1,bi2]: C [ao1], D [ao2], E [bo1]:
F [ao1, co1][fo1], G [do1][go1, go2],
H [eo1, bo2][ho1] } ----------(14)

그림 8과 같이 혼합적으로 결합되는 새로운 소프트웨어 칩 R의 입력은[ai1, ai2]와 [bi1, bi2] 이고, 출력은 [fo1], [go1, go2], [ho1]인 기본 소프트웨어 칩과 같으므로 정리 1에 의하여 표현식 (14)는 표현식 (15)와 같이 표현할 수 있다.

{ R [ai1, ai2, bi1, bi2][fo1, go1, go2, ho1] }
----------(15)

병렬로 결합된 하나의 단위 소프트웨어 칩이 같은 형태의 소프트웨어 칩과 직렬적인 구조와 결합될 때 그림 9와 같이 입력과 출력이 엇갈려서 결합되는 경우가 있다.

그림 9와 같은 구조는 정의4~ 정의7 및 정리 2에 의하여 표현식 (16)과 같이 표현한다.

R = { A [ai1, ai2], B [bi1, bi2]:
C [ao1, bo1][co1, co2], D[ao2, bo2][do1, do2] } --(16)

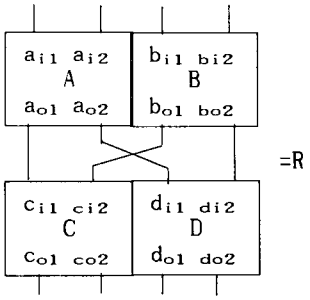


그림 9. 엇갈린 혼합 결합
fig. 9 crossed mixing composition

그림 9와 같이 결합되는 새로운 소프트웨어 칩 R의 입력은 [ai1, ai2]와 [bi1, bi2]]이고, 출력은 [co1, co2] 와 [do1, do2]인 기본 소프트웨어 칩과 같으므로 정리 1에 의하여 표현식 (16)은 표현식 (17)과 같이 표현할 수 있다.

{ R [ai1, ai2, bi1, bi2][co1, co2, do1, do2] }
----------(17)

Ⅳ. 비교 및 분석

재사용되는 소프트웨어 모듈을 표현하는 방법을 Diaz와 Lenz및 Browne이 제안한 것과 본 논문에서 제안한 것의 특성을 비교하면 표 1과 같다.

표 1. 재사용 모듈 표현 방법의 특성 비교

종류	기술방법	표현 대상	적용 분야
Diaz	프로시저 형식	필요한 소프트웨어 모듈	가장 근접한 모듈검색
Lenz	프로시저 형식	필요한 소프트웨어 모듈	가장 근접한 모듈검색
Browne	표현식	부품화한 소프트웨어 요소	소프트웨어 부품의 결합
본논문	표현식	부품화한 소프트웨어 요소	소프트웨어 부품의 결합

표 1에서와 같이 Diaz와 Lenz가 제안한 방법은 필요한 기능을 프로시저로 기술하고, 가장 가까운 모듈을 찾아내는 방법이므로 직접 모듈을 결합시키는 데에 사용하기는 불편하다.

Browne과 본 논문에서 제안한 방법은 소프트웨어를 부품화 시켜서 필요한 곳에 직접 결합할 수 있는 방법이

며, 이 2가지 방법에 대한 세부적인 적용 분야는 표 2와 같다.

표 2. 소프트웨어 부품 결합 표현 방법의 비교

구분	직렬 결합	병렬 결합	복합 결합	반복 구조	다중 결합	혼합 결합	흐름 표시
Browne	가능	가능	가능	불가	불가	불가	불가
본 논문	가능	가능	가능	가능	가능	가능	가능

표 1과 표 2에서 나타난 것과 같이 본 논문에서 제안하는 방법은 부품화된 소프트웨어 부품 결합과 분리에 대한 표현을 보다 구체적이고 다양하게 표현할 수 있다는 것으로 입증된다.

V. 결론

소프트웨어 병목현상을 해결하기 위하여 프로그래머의 생산성을 향상시키기 위한 시도는 오래 전부터 연구되어 왔지만 확실한 방법은 개발되지 못하였다.

소프트웨어 재사용 방법은 개념이 간단하고 공감이 가는 분야이지만 너무나 많은 작업을 필요로 하는 실행에 어려움이 뒤따라 목적한 만큼의 성과를 거두지 못하고 있다.

소프트웨어 칩을 사용한 소프트웨어 결합 방법은 소프트웨어 각 컴포넌트를 하드웨어 칩에 비유하여 칩들을 조립하여 새로운 상품을 만드는 것과 같이 소프트웨어 칩을 결합하여 새로운 소프트웨어가 되도록 하는 방법이다.

이 방법은 각 컴포넌트가 독립적으로 관리되기 때문에 유지보수가 편리할 뿐만 아니라 어떠한 목적에도 그 칩이 필요하다면 사용할 수 있는 장점이 있다.

소프트웨어 칩을 사용할 때는 모든 기능이 칩 속에 캡슐화 되어있고 사용자는 단지 결과만 사용하기 때문에 융통성이 작다. 그러므로 변화가 적은 분야에 대한 업무가 알맞으며, 칩들을 연결해주는 인터페이스는 표준화가 요구된다.

소프트웨어 칩들을 결합하는 중계 역의 기능을 표준화된 프로토콜을 사용한 소프트웨어 버스로 대응시키고 모든 컴포넌트들이 이곳에 결합되어 네트워크 제어기에 의하여 제어되면 프로그래머는 소프트웨어 버스와 컴포넌트 연결에 대한 지식만 있으면 효과적으로 칩들을 결합시킬 수 있게된다.

표준화된 인터페이스 경로를 통하여 결합된 소프트웨어 칩에서는 데이터 종속성만을 표현하여 칩들을 결합할 수 있으며, 이들은 직렬뿐만 아니라 병렬로도 결합이 가능하며, 이를 응용하여 복합적인 결합도 가능하다.

소프트웨어 칩에 의한 재사용 방법은 표준화 가능한 컴포넌트가 많이 있는 경우에는 매우 효과적인 재사용방법이라고 보여진다.

이러한 소프트웨어 칩들이 결합되는 관계는 본 논문에서 보여주는 표기법으로 직렬, 병렬 및 혼합된 형태와 재귀적인 관계를 확실하게 표현할 수 있다.

앞으로의 과제는 더 많은 컴포넌트의 표준화가 이루어져야 하겠고 보다 간편한 프로토콜의 표준화 및 대중화가 이루어져야 하겠다.

참고 문헌

[1] Ellis Horwitz and John B. Munson, "An Expansive View of Reusable Software", IEEE Tran. SE, Vol. SE-10, No. 5, PP.477-487, september, 1984

[2] J. C. Browne, Teajae Lee and John Werth, "Experimental Evaluation of Reusability Oriented Parallel Programming Environment" IEEE Trans. on Software eng., Vol. 16, No. 2, pp 111-120, Feb. 1990

[3] Lamar Ledbetter and Brad Cox, "SOFTWARE-ICs", BYTE, Vol. 9, NO. 6, PP. 28-36, July 1987

[4] Manfred Lenz, Hans Albrecht, and Peter F. Wolf, "Software Reuse through Building Block", IEEE Software, PP. 34-42, July 1987

[5] Michael F. Korns, "Reason and the Software Bus", BYTE, Vol. 9, NO. 1, PP. 28-36, June 1985

[6] Robent G. Lanergan and Chanles A. Grasso, "Software Engineering with Reusable Designs and Code", IEEE Trans. Software Eng. , vol. SE-10, NO. 5,

PP. 498-501, september 1984
[7] Ruben Prieto Diaz and Peter Freeman, "Classifying Sofftware for Reusability", IEEE Software, PP. 6-16, January 1987
[8] Scott N. Woodfield, David W. Embly and Del T. Scott, "Can programmers Reuse Software?", IEEE Software, PP. 52-59, July 1987
[9] Taejae Lee and J.C. Browne, "Intersection of Paralled Structuring and Reuse of Software component ", International conference on Parallel Processing 1989.
[10] Taejae Lee and J.C. Browne, "Experimental Evaluation of a Reuseability Oriented Parallel Programming Environment", submitted to IEEE Trans. Software Eng.
[11] T. Capers Jones, "Reusability in programming: A Survey of the State of the Art", IEEE Trans. Software Eng., vol. SE-10, NO. 5, PP. 488-493, september 1984
[12] Ted J. Biggerstaff, "Foreward" IEEE Trans. on Software Eng., Vol. SE-10, No. 5, pp. 474-476, Sept. 1984
[13] Ted Bigggenstaff and Charles Richer, "Reusability Framework Assessment and Directions", IEEE Softwarem, PP. 41-48, March 1987
[14] Will Tracz, "Software Reuse : Motivators and Inhibitors", Proceedings of COMPC-ONS'87, PP. 358-368, 1987
[15] Will Tracz, "Software Reuse Myths ", ACM SIGSOFT Software Engineering Notes, Vol. 13, NO. 1, PP. 17-21, January 1988

저 자 소 개

김 홍 진
광운대학교 전자계산학과 이학 석사 및 이학 박사
1998 경원전문대학 컴퓨터정보과 교수 재직중
1974년 한국전자계산소
1975년-1983년까지 (주)민컴에 근무
한국OA학회 총무이사를 역임
전국 대학정보전산기관협의회 상임이사 역임
현재 한국OA학회 부회장
전국 대학정보전산기관협의회 감사
한국 컴퓨터산업교육학회 이사