병렬처리 소프트웨어의 복잡도 측정

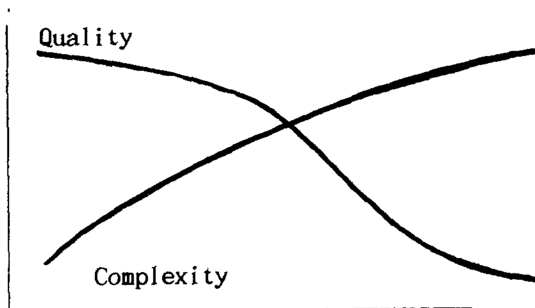
배 정 미(대경전문대학 전자계산학과)

제 1 장 서 론

소프트웨어 시스템이 대형화 됨에 따라서 소프트웨어의 개발, 유지 비용의 감소와 품질 보증(quality assurance)에 큰 관심을 갖고 연구가 진행되고 있으며 그 연구과정으로는 소프트웨어 라이프 싸이클과 소프트웨어 제품을 분석하여 생산 비용과 품질에 영향을 주는 요소를 측정하려는 시도를 들 수 있다. 특히 소프트웨어 공학에서 소프트웨어 품질의 정량적인 측정(quantitative measure)방법으로 복잡도에 관한 연구는 날로 증대되고 있다. 소프트웨어의 복잡도(Complexity)란 컴퓨터 프로그램을 개발하고 유지하는데 소요되는 노력과 비용에 영향을 주는 소프트웨어의 특징요소 즉, 데이타 구조의 형태, 중첩(Nesting)된 정도, 조건분기의 수, 인터페이스(Interface)의 수 등의 복잡한정도를 일컫는다. 이러한 정량적인 측정의 결과를 소프트웨어 라이프 싸이클에 적절히 활용한다면 소프트웨어 공학의 목표인 품질 향상과 비용감소에 큰 진전을 가져다 줄 것이다. 예를 들면 소프트웨어의 복잡도 측정으로 Software생산 주기의 개발 cost의 67%를 차지하는 유지 보수 단계의 비용을 절감 할수 있다. 유지 보수단계의 복잡도 정도는 이해의 정도, 수정용이성, 시험 용이성을 내포하고 있다.[1] 즉 복잡도는 소프트웨어의 품질 향상과 비용감소에 밀접한 상관 관계를 가진다.[1][9]

그림[1]은 소프트웨어의 복잡성과 소프트웨어의 품질과의 상호 관계를 나타내고 있다.

<그림 1> 소프트웨어 품질과 복잡도의 상호관계



또한 측정된 복잡도 메트릭스(Metrix)는 개발될 software제품을 평가 하는 기준이 될뿐만 아니라 프로젝트에서의 스케줄링(Sceduling), 비용 산정(Cost Estimation)을 가능하게한다.(품질 평가, 프로젝트 관리 측면)[3] 그결과 사용자와 시스템 개발자,

경영 관리자의 요구사항에 대하여 분석 결과를 정량적인 복잡도 수치로 제시(Quantitative measure)하여 줌으로서 이들과 시스템 분석가(System Analyst) 상호간의 이해정도를 높이고 개발 목적에 맞는 정확도 높은 소프트웨어를 생산 할수 있도록 지원 할 수있도록 한다. 그러나 기존의 복잡도 측정에 관한 연구는 주로 간단한 소프트웨어(Sequential processing)에 적용하기 쉬운것으로서 순차적이고, 하나의 프로세스에 집중된 소프트웨어에 적용할 수 있는 측정방법[20]으로 제안되었으며 시스템을 병렬적으로 처리해 줄 수 있는 병렬처리 시스템의 복잡도 측정에는 부적합한 방법이다.

병렬처리 시스템은 여러개의 하드웨어 프로세스로 이루어져 있으므로 이 시스템에서 수행되는 소프트웨어는 여러 개의 프로세스에 분산되어 수행될 수 있는 소프트웨어이어야 한다. 이러한 소프트웨어를 병렬처리 소프트웨어라 하는데, 이것은 공통된 일을 처리하기 위하여, 내부적으로 연결된 여러개의 컴퓨터(하드웨어 프로세스)에 할당되어진 고정된 수의 소프트웨어 프로세스 집합이다. 내부적으로 연결되어진 컴퓨터들은 메시지를 통하여 연결되므로, 소프트웨어 프로세스들 사이의 통신 또는 메시지를 통하여 이루어진다. [10][13] 이러한 병렬처리 시스템에 있어서 가장 중요한 문제는 정확하고 신뢰할 수 있는 병렬처리 소프트웨어를 개발하는 것이다. 현재 병렬처리 소프트웨어 개발에 관한 연구가 설계, 구현, 검증등 모든 분야에서 활발히 이루어지고 있으며 대표적 병렬처리 프로그래밍 언어로서는 에이다(Ada), CSP, Occam언어, Concurrent C 등을 들수 있다. 이러한 병렬 소프트웨어의 복잡도 측정은 병렬처리 소프트웨어의 개발 비용및 보수 유지 비용을 감소시키고, 품질 보증과 신뢰성 향상을 위하여 프로그램을 객관적이고 정량적으로 측정할수 있는 평가 기준이 된다. 따라서 분산 소프트웨어 복잡도 측정방법에 대한 연구가 절실히 요구 되어진다. 그러나 병렬처리 소프트웨어 복잡도 측정에 관한 연구는 아직 미비한 실정이다.

본 연구에서는 프로세스간의 통신, 프로그램 모듈화에 촛점을 맞추어 병렬처리 소프트웨어의 복잡도 측정 방법을 제안한다. 우선 병렬 처리가 가능한 대표적이 프로그램 언어인 에이다(Ada)언어의 특징을 분석,연구하고 에이다 언어에서 나타나는 병렬성을 페트리네트로 모델링하여 병렬처리시 영향을 미치는 복잡도 형성 요인을 분석하고자 한다.

제 2 장 기존 연구내용

1. Program Size에 기반을 둔 측정방법

1) COCOMO MODEL..LOC(Line Of Code 측정 방법)[2]

2) Halstead의 Software Science[4][5]

: Program의 Operator와 Operand를 기반으로 Program의 Length, Program의 Level 등과 같은 성질을 예측하는 함수를 유도한다. 이때의 최종 결과 함수는 노력(Effort) 수치로서 심리적 복잡도(Psyclological Complexity)를 나타내며 Program을 작성하는데 드는 노력과 작성된 Program을 이해하는데 드는 노력으로 Program error수에 밀접한 관계를 갖는다.

2. Program 제어 흐름에 기반을 둔 측정방법

1) Mcacabe의 Cyclomatic Number[6]

: Program내에 존재하는 제어분기의 수나 제어분기간의 구조적 특징을 측정하는 방법(Testing과 관련 있음)이다. 이방법은 그래프(Graph)이론에 근거한 복잡도 측정 방법으로서 Program이주어지면 이를 제어그래프(Control graph)로 변환시킨다.
제어흐름이 순차적인 블럭(block)은 하나의 노드(node)에 대응되고, 각 분기(branch)는 하나의 edge에 해당된다.

(G) = e - n + 2p

- e : edge
- n : node
- p : graph에서의 연결 요소의 수

V(G)의 값이 커질수록 개발시간, 비용, 그리고 error가 발생할 확률이 커지는것으로 나타났다.
이 기준은 control flow만 고려 하였지 자료흐름을 고려 하지 않았다.

3. Data구조, 흐름(Data structure & flow)에 기반을 둔 측정방법[7][8]

Program내의 자료(Data)들이 어떻게 사용되고 구성되며, 활용되는지에 의해 복잡도를 측정한다.

1) Henry & Kafura의 Information flow[7]

= Length * (Fan-in * Fan-out)^2

이때 Length란 source code의 line 수

4. 혼합적 복잡도 측정 방법

:Program의 크기(Size), 자료구조, 제어흐름등 software의 복잡도에 영향을 미치는 특성들을 2가지이상 고려 함으로써 보다 신뢰성있는 특정을 하기위해 제안된 방법이다.

1) Hansen의 복잡도 측정

McCabe의 Cyclomatic Number와 Halstead의 Software Science를 같이 고려한 측정 방법.

2) Ovido의 Program복잡도 model

제어흐름과 Program크기를 고려한 척도.

제 3 장 페트리네트 모델

페트리네트는 동시적(concurrent)이고 비동기적(asynchronous)인 시스템의 모델링과 분석을 위한 훌륭한 그래픽 도구들중의 하나이다. 페트리네트 모델의 이용분야는 성능 분석 및 통신 프로토콜(Protocol)분야뿐만 아니라 분산 소프트 웨어, 시스템 분석, 병

렬처리 및 병렬처리 프로그램, 이산사건 시스템, FMS(Flexible Manufacturing system)등의 분야에 이용되고 있다.[21]

1. 페트리네트의 기본 구조

페트리네트의 기본 구성요소는 플래이스, 트랜지션, 입력, 출력 함수로구성되어진다.

- 페트리네트 의 구성요소 = (P,T,I,O)
  - P: 플래이스의 유한집합
  - T: 트랜지션의 유한집합
  - I: 트랜지션에서 플래이스로 입력되는 입력함수
  - O: 플래이스에서 트랜지션으로의 출력함수

2. 표기형태

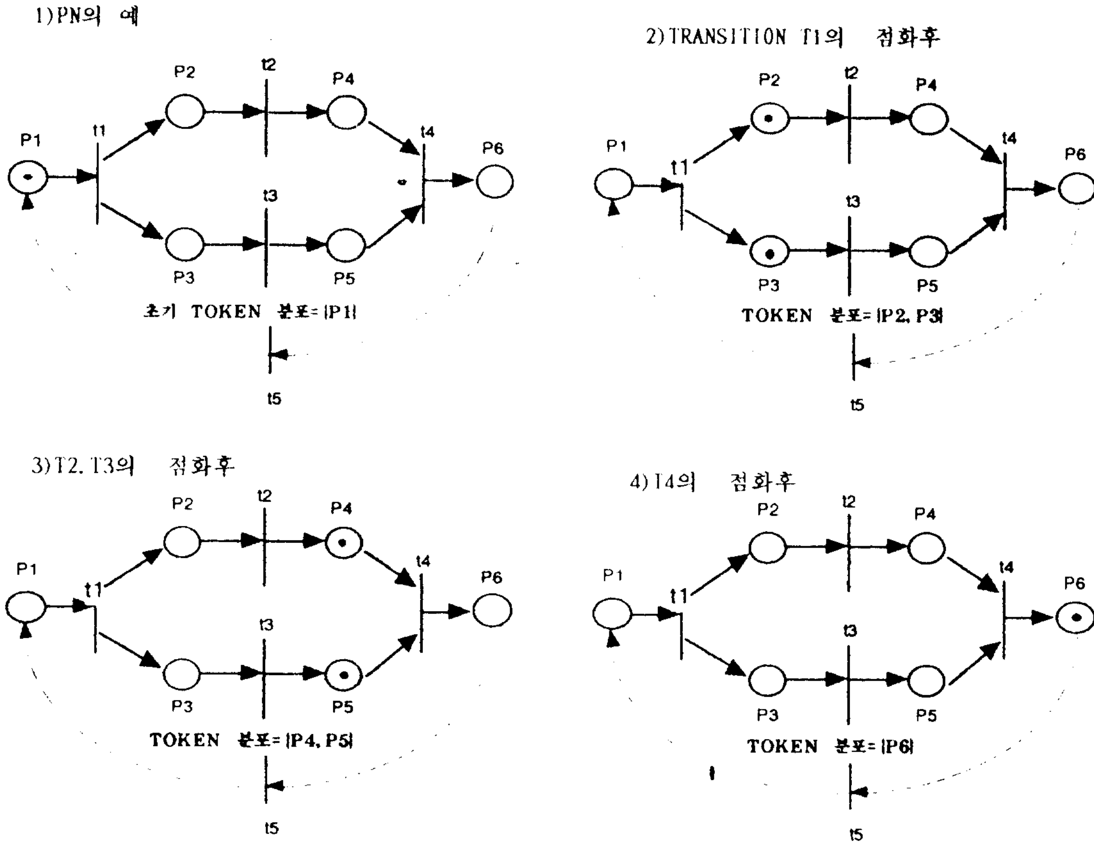
플래이스는 원으로 표시하고 트랜지션은 막대로 표기되며 플래이스안의 검은 점은 토큰의 분포 상태를 나타낸다. 입,출력 함수는 화살표로서 표기된다.

3. 정의

- 점화 가능하다(Firing Enable)는 것은 어떤 트랜지션(Transitin)의 입력 플래이스(Place) 모두에 토큰(Token)이 존재할때 이 트랜지션은 점화 가능 하다고 한다.
- 점화(Firing)란 어떤 트랜지션 이 점화가능 할때 그 트랜지션이 점화되면 그 트랜지션 입력 플래이스에 있던 토큰들이 1개씩 출력 플래이스로 옮겨진다.(동시에 여러 트랜지션이 점화가능하면 임의로 한개 택하여 점화한다.) -> 페트리네트의 토큰분포가 변하게 된다.
- 교착상태 발생은 PN내에 점화되지 않는 트랜지션이 있을때 교착상태(Deadlock)가 발생하였다고 한다.
- 바운드니스(Boundedness)라는 것은 토큰(Token) 의 수가 무한히 증가하지 않는다면 바운디드(Bounded)하다고 한다. 예를들어 A플래이스내의 토큰의 수가 k정수개를 초가 하지 않으면 이때의 플래이스는 K 바운디드하다고 한다.
- 자원의 안전성(Sefeness)이란 플래이스마다 토큰이 1개 까지만 존재 할수 있다면 세이프(Safe)하다고 한다. 안정성은 바운디니스의 특별한 경우이다.
- 보존성 (Conservation)이란 자원을 나타내는 토큰들이 생성되거나 소멸되지않는 성질을 일컫는다. 이는 상호 배제의 문제를 해결한다.
- 생동성(Liveness)이란 트랜지션이 교착상태에 빠지지않았음을 일컫는다.
- 도달 가능성(Reachability)이란 시스템내의 교착상태의 유무를 판별하는데 필요하며 또한 시스템의 원상회복이 가능한지도 판단 할수있다.

그림[2]는 PARBEGIN , PAREND를 표현하는 페트리네트 모델링의 예이다.

<그림 2> 페트리네트 모델



4. 페트리네트의 분석방법

일반적으로 패트리네트를 이용하여 어떤 시스템을 모델링하는 방법은, 우선 그 시스템을 반영하는 페트리네트을 그리고 그 페트리네트의 성질들, 즉 도달성(Reachability), 생존성(Liveness)과 자원의 안전성(Safeness)과 기타 성능평가등을 한 후, 만족스럽지 못하면 재차 페트리네트를 수정하고, 완전하면 그것을 실제로 프로그래밍 한다. 여기서 자원의 보존성은 무한히 자원이 필요하게 되는지의 유무를 판별할 수 있고, 도달성 또는 생존성 문제는 시스템내에 교착상태 유무를 판별하는데 필요하며 또한 시스템의 원상회복이 가능한지도 판단할 수있다. 이 과정에서 사용되는 페트리 네트의 분석기법으로서는 REACHABILITY TREE방법과 MATRIX EQUATION 방법등이

있다.

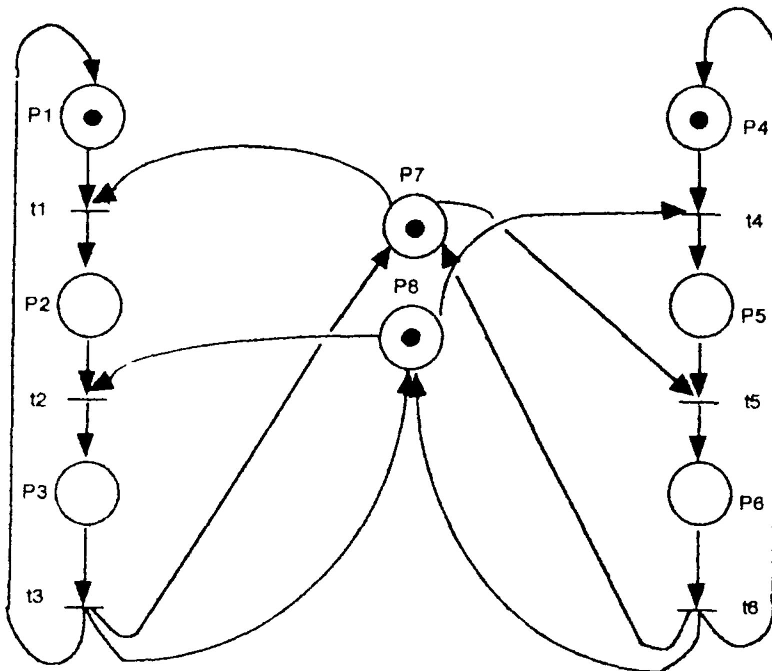
REACHABILITY TREE는 페트리네트의 점화(firing)에 따라 계속 변화되는 토큰의 분포상태를 표현한다. 이때의 분포상태란 토큰의 분포상태를 말하며 트랜지스트는 페트리네트의 점화를 나타낸다. 그러나 이방법으로는 도달성 문제를 해결할수 없고, 자원의 보존 문제는 분석이 가능하다. 또한 페트리네트의 복잡성에 따라 페트리네트의 기하급수적 증가를 초래하여 많은 분석시간이 요구되는 단점이 존재한다.

MATRIX EQUATION 방법은 페트리네트를 행열 D로 표시한 후, $M = M_0 + D^{*}x$의 해를 구하는 방법이다. 여기에서 D는 $D^{+} - D^{-}$ 이며, D^{+} 는 입력되는 place의 수를 나타낸 행열 이며 D^{-} 는 입력되는 플래이스의 수를 나타낸 행열이다. M_0는 초기 토큰분포 벡터(vector)를 나타내며 M은 원하는 token분포 벡타 이다. $M = M_0 + D^{*}x$의 해를 구하면 이페트리네트는 도달성이 있는것이고 못구하면 도달성이 없다. 이방법은 행열 D가 원래의 PN의 구조를 반영하지 못하며 점화의 중간 순서를 알수 없으며 도달성 문제 해결의 필요조건일 뿐이고 충분조건이 되지 못하는 단점이 존재한다.

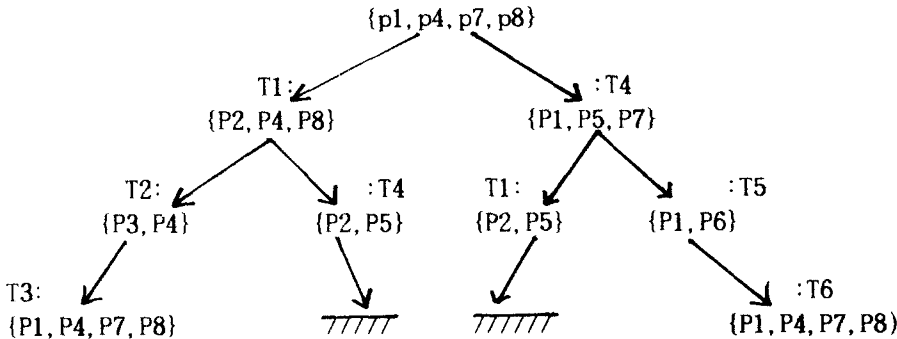
그림[3]은 페트리네트 모델의 분석과정의 예이다.

〈그림 3〉 페트리네트의 분석

1) 페트리네트 모델



2) RECHABILITY TREE방법



여기서, 토큰 분포가 {P2, P5}일때 교착상태가 발생하며 각 플래이스내에 토큰이 1개 이하이므로 안정성(SAFE)이 보장된 모델이다. 또한 초기 토큰 분포{P1, P4, P7, P8}으로 회복이 가능함을 알수 있다.

3) MATRIX EQUATION방법

D^{+} =		D^{-} =		D는 $D^{+} - D^{-}$ =
	01000000		10000010	-1 1 0 0 0 0 -1 0
	00100000		01000001	0 -1 1 0 0 0 0 1
	10000011		00100000	0 -1 1 0 0 0 0 -1
	00001000		00010001	0 0 0 0 -1 1 -1 0
	00000100		00001010	0 0 0 0 -1 1 -1 0
	00010011		00000100	0 0 0 1 0 -1 1 1

M_0 = ⟨10010011⟩, M=⟨10010011⟩, M = M_0 + $D^{*}x$에서 x의 해가 존재 않는다. 그러므로 초기 토큰분표 M_0는 다시 회복될 수있다. 그러나 M_0는 M으로 도달가능하다.(t1->t2->t3, t4->t5->t6로 점화)

제 5 장 에이다 언어(ADA LANGUAGE)의 기본특성

Ada 언어는 시스템 프로그램 응용, 실시간 및 병렬처리 응용등을 위한 프로그래밍 언어로써 1970년대 말에 설계되어 1980년대 말에 들어와서야 전체 Ada가 PC에 구현되었으며 사용자에게 Ada 프로그래밍을 작성하고 사용할 수 있는 환경은 설계 후 비교적 늦게 조성되었다. 1970년대 초에 개발되어 그 이후 거의 모든 새로운 언어에 커다란 영향을 준 Pascal 언어로부터 Ada언어는 많은 영향을 받아왔다. 그러나 Pascal 언어에서 직접 영향을 받았다기 보다는 Pascal과 SIMULA의 영향을 많이 받아 설계된 Modular, Concurrent Pascal, Mesa, Euclid, CSP등 최근언어들의 개념을 수렴하여 Ada 언어와 CHILL언어가 설계되었다고 볼수있다.

Ada언어는 미국방성의 지원으로 설계되었는데 군사 목적의 명령및 제어 시스템은 여러개의 병행 활동을 감시하고 또한 이들 감시되는 활동들의 예측할 수 없는 변화에 신

속히 대처해야하는 일들이다. 이들 시스템을 구성하는 컴퓨터에 사용될 효율적인 소프트 웨어를 작성하는 작업은 병행 프로그래밍 언어를 사용함으로써 대단히 편리하게 되었다. 이것이 바로 미국정부가 에이다(Ada)의 개발을 지원한 이유이다.

Ada언어는 병렬실행의 특성과 더불어 Package의 프로그램 단위별 개념이 강하다고 볼수 있으며 프로그램의 안전도, 유지보수, 그리고 인간의 활동과 효율에 중점을 두고 있다고 할수 있다. Ada프로그램은 구조화된 프로그램단위로 구성되어 있는데 프로그램 단위로는 부프로그램, 패키지(Package), 태스크(Task)등을 들수 있으며 이들은 선언부이거나 실행부가 될수도 있으며 내부적으로 여러가지 구조를 포함할수가 있다.

1. 순차처리 형의 문장구조(Structured Statement)

타 순차적 프로그램 언어와 같이 선택문과 반복문이 지원되는데 선택문에는 IF문,CASE문을 들수 있으며 반복문에는 세가지 LOOP반복문을 제공하는데 FOR문, WHILE문, LOOP반복문 내의 임의 장소에서도 조건이 맞으면 반복 수행을 끝낼수 있도록 하기 위한 EXIT문을 제공한다.이는 PASCAL의 REPEAT문의 사용과 동등하다.[22]

그림[4]는 순차적 형태의 문장 예이다.

<그림 4> 순차처리문

(1) IF문

```
if N<0 then COUNT:=COUNT+1;
        else BAL:=BAL+1;
   end if
```

(2) CASE문

```
 case command(i) is
        when '+'|'-' => sign:=true;
        when '0'..'9' => digit:=true;
        when others => no-digit:=true;
   end case;
```

(3) FOR문

```
   for i in firstweek'range
       loop
       firstweek(i):=false;
       end loop
```

(4) WHILE문

```
   current:=head;
   while current /= null
    loop
     ...
     current:=current.next;
```

```
    end loop;
```

(5) LOOP-EXIT문

```
    loop
    i:=i+1;
    exit when sentence(i) /= '$';
   end loop;
```

2. 병렬처리형의 문장구조

Ada에서는 여러개의 독립된 활동들이 프로그램 작성을 허용하고 있다. 이러한 활동들을 태스크(task)라고 하며 여러개의 태스크들은 병행(Concurrent)실행을 시킬 수 있는 기본 프로그램 단위이다. 태스크끼리는 병행으로 실행되며 명세부(Specification part)와 몸체(Body)를 가지고 있는데, 명세부에서는 사용자가 태스크를 통하여 사용가능한 자원(Resources)들을 명세하게 되며, 몸체부에서는 이들 자원의 구체적인 구현을 정의한다. 이 태스크 동기화는 랑데뷰(Redezvous)라 불리는 기법에 의해 수행된다. 호출자(Caller)태스크가 다른 태스크의 진입점(Entry)을 호출할 수 있으며 서버(Server)태스크가 자기의 진입점중 하나에 대한 호출을 받아들이기 위해 accept문을 수행한다. 호출이 받아들여지면 랑데뷰가 발생한다. 호출자 태스크는 전입점 호출 내의 인수(Parameter)를 이용하여 서버(Server)에게 데이타를 전달한다. 결과 역시 호출시의 인수를 통하여 전달된다. 호출자 태스크는 서버가 처리하는 동안 대기한다. 그림[5]는 Ada언어에서의 태스크 명세와 몸체에 대한 예이다.[22][23]

<그림 5> 에이다 언어의 TASK

- 태스크 명세(TASK SPECIFICATION)

```
task MAILBOX is
    entry SEND(INMAIL: in MESSAGE);
    entry RECEIVE(OUTMAIL: out MESSAGE);
end;
```

- 태스크 몸체(TASK BODY)

```
task body MAILBOX is
  BUFFER:MESSAGE;
begin
   loop
     accept SEND(INMAIL:in MESSAGE) do
        BUFFER:=INMAIL;
     end;
     accedpt RECEIVE(OUTMAIL:out MESSAGE) do
        OUTMAIL:=BUFFER;
     end;
   end loop;
end MAILBOX;
```

제 5 장 병렬처리 소프트언어의 복잡도

본 논문에서는 병렬처리 프로그램 언어의 예로서 에이다 프로그램구조를 이용하여 복잡도 형성요인을 분석하여 복잡도 측정치를 산출하고자 한다. 에이다 프로그램에서는 여러개의 순차적 프로세스들이 병렬적으로 수행되다가 서로 간의 통신이 필요할 때는 메시지를 보냄으로서 두 프로세스의 동기화가 이루어진다. 본논문에서는 에이다 프로그램의 복잡도를 측정하기위하여 이러한 특징을 고려하여 상호 연결되어 컴퓨터 상에서 수행 될 수 있는 프로그램의 모듈과 각 프로세스 간의 랑데뷰를 분석하고자 한다. 제안하는 에이다 언어에서의 복잡도 형성요인으로서는 다음과 같이 나누어 볼 수 있다. 첫째 요인은 병렬처리가 가능한 프로세스의 수(NUMBER OF TASK)로서 TN으로 표시하며 두번째 요인으로는 순차적 처리가 이루어지는 각 모듈내의 복잡도(COMPLEXITY OF MODULE)로서 CM으로 표기한다. 마지막 세번째 요인은 병렬 처리를 행하는 프로세스들 사이의 메시지 전달(Message Passing)을 표현하는 인터페이스 요인(COMMUNICATION CHANNEL)으로서 CC로 표기하며 이렇게 3가지 요인으로 나누어 고려하고자 한다.

에이다프로그램에서의 PARALLEL PROCESSING이 가능한 PROCESS의 수는 태스크 단위로 병렬 처리가 이루어지므로 태스크의 수를 측정 하도록하며 모듈의 복잡도는 에이다프로그램에서 모듈의 개념으로 볼수있는 각각의 프로시듀어, 함수, 페키지 , 태스크내에서이루어지는 순차적처리 복잡도를 측정하여 합한값을 구하도록한다. 이때의 순차적 복잡도 측정은 기존에 제안된 프로그램 크기에 기반을 둔 측정방법, 프로그램 제어흐름에 기반을 둔 측정방법, 데이타 구조,흐름에 기반을 둔 측정방법, 혼합적 복잡도 측정방법등에 적용하여 측정 한다.
마지막으로 인터페이스 복잡도는 에이다a프로그램을 페트리네트로 변형한후 태스크간의 통신 복잡도요인을 분석한다. 페트리네트를 이용하여 통신 복잡도를 계산하는 장점으로는 프로그램을 보기 쉽게 모델링화하여 좀 더 이해도를 높이고 알기 쉽게 측정 하기위해서이다. 그림[6]은 병렬처리를 지원하는 태스크를 나타내는 Ada프로그램의 일부분이며 그림[7]은 Ada 프로그램을 페트리네트로 변형한 예이다.

그림[7]에서 점선부분은 프로그램상에서 연결된 다른 문들을 의미한다. Place P6는 loop구조를 나타내며 t3, t4, p5, p2, p3, p4는 엔트리 호출을 의미하며 t5, t6, p7, p2, p3, p4는 ACCEPT문을 표현한다. 그리고 p2, p3, p4플래이스들은 TASK2 에서의 진입점 ENTRY2에대한 인터페이스(Communication channel)로 볼수 있다. p2는 상호배제를 위한 초기마킹 플래이스가 되며 TASK2.ENTRY2문에 의하여 t3가 점화되며 t3가 성공적으로 점화되면 p3, p5플래이스가 활성화된다. 이는 랑데뷰의 초기화 작업이며 ACCEPT문의 시작에 해당된다. 이때의 p3 플래이스는 Sending플래이스가 된다. p4플래이스는 Acknowledging플래이스가 된다. ACCEPT문이 끝나게되면 t4가 점화된다. 여기에서 Communication channel이 되는 p2, p3, p4의 발생수는 통신 복잡도의 중요 요인으로 고려되어진다.

따라서 대표적인 병렬처리 언어인 Ada 언어의 총복잡도는 다음과 같이 계산되어질수 있다.

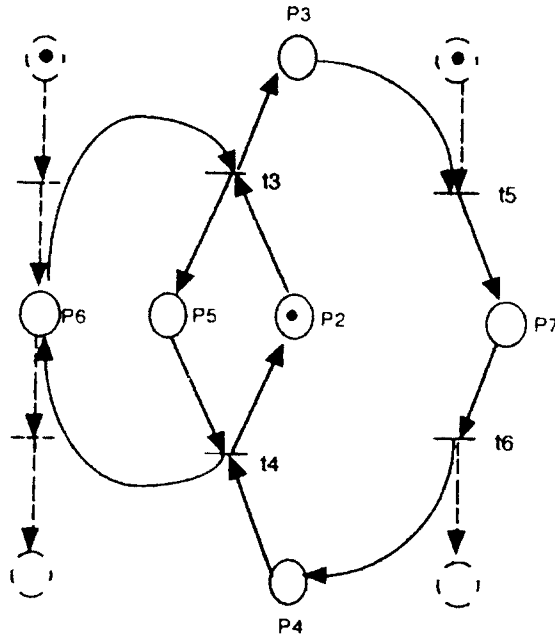
$$\text{총복잡도} = \left(X * \sum_{i=1}^{n} PN_i \right) + \left(Y * \sum_{j=1}^{n} CM_j \right) + \left(Z * \sum CC \right)$$

그림[6] TASK의 사용예

```
task body TASK1 is
begin
  while cond1 loop
        TASK2.ENTRY2;
  end loop;
end TASK1;


task TASK2 is
  entry ENTRY2;
end TASK2;
task body TASK2 is
begin
  accept ENTRY 2;
end TASK2;
```

그림[7] 페트리네트 모델로의 변형



이때의 X,Y,Z는 총복잡도에 미치는 각 요인의 영향을 가중치로 표시한것이며 이 가중치값의 계산은 실제 프로그램을 시뮬레이션한후 통계치를 구하여야 하며 본논문에서는 생략한다.

제 6 장 결론

최근 다중 프로세서 컴퓨터의 발전이 가속화 됨에 따라 이들의 능력을 충분히 활용할 수 있도록 다중 프로세서의 병행 계산 능력을 최대한 이용할 수 있는 새로운 병렬처리 개념의 프로그래밍 언어가 개발되고 있다. 병렬처리 소프트웨어 시스템의 개발과 유지 보수는 비용면에서나 시간적인 측면에서 더 많은 노력을 요구한다. 본 연구에서는 에이다 언어의 병렬 실행구조를 페트리네트로 모델링하여 복잡도 형성요인을 분석하여 병렬 처리 복잡도라는 정량적인 측정치를 제시하여 시스템의 개발 비용 산정, 유지 보수와 신뢰성 향상을 지원하고자 하였다. 앞으로의 연구과제로는 총복잡도를 구성하는 각각의 복잡도 요인의 가중치 산정에 대한 연구가 과제로 남아 있다.

지금까지의 복잡도연구는 대부분이 구현단계 이후 생성되는 쏘스 코드(Source code)를 대상으로 이루어졌기 때문에 분석단계에서 요구 되어지는 소프트웨어의 비용을 산출하고자 할 때 별 가치 없는 방법이 되어 버린다. 따라서 지금까지 분석단계에서 구조적 분석도구로 사용되어지는 DFD(Data Flow Diagram)[24] 다른 활용분야로서 모듈 복잡도 측정에 이용하는 방법에 대한 연구를 과제로서 남겨둔다. 즉,DFD를 작성한 다음 복잡도를 측정하게 된다면 설계될 시스템의 기능성을 향상시키게 된다.즉 과다하게 가중된 프로세스(Process)의 복잡도를 추정함으로써 잘못 정의된 프로세스를 분할(Partition)하여 재분석하도록 유도할수있다.