論文94-31B-11-1

두레 : 분산시스템을 위한 병행연산모델

(DOORAE : A Concurrent Computation Model for Distributed Systems)

金 大 權*, 朴 忠 植**, 李 林 建*, 李 容 錫*, 朴 圭 泰*

(Dae Gwon Kim, Choong Shik Park, Im Geun Lee, Yong Surk Lee and Kyu Tae Park)

要 約

문제의 병행처리를 위한 모델링 방법과 문제의 병행성을 동적 환경에서 자동으로 검출하기 위하여 병행연산모델 두레를 설계하고 두레언어 DL로 구현하였다. 두레모델은 문제의 모델링과 연산수행을 위해 단순하고 일관된 개념의 객체 정의와 메시지 전송개념을 지원한다. 문제의 병행처리를 프로그램에 명시하지 않고도 암시적으로 지원할 수 있도록 병행성의 검출 기준을 제안하였으며, 동적 환경에서 병행객체를 생성하여 최대한의 병행성을 보장하도록 하였다. 또한 객체의 연산 진행을 효율적으로 보장해 줄 수 있도록 Waiting Variable의 개념을 제안하였다.

Abstract

A concurrent computation model Doorae and its description language DL are developed to model problems of parallel and distributed systems. Doorae model has simple and uniform concepts of object and message passing for problem modeling and computation. A method for detecting parallelism implicitly, with no exact description of parallelism in program, is proposed. Furthermore, the method assures the maximum parallelism in dynamic environment by creating concurrent objects. Also a concept of Waiting Variable to insure maximum computation efficiency of objects is proposed.

*正會員, 延世大學校 電子工學科
(Dept. of Elec. Eng., Yonsei Univ.)
**正會員, 永同工科大學 電算學科
(Dept. of Computer Science, Young-dong Institute of Technology)
接受日字 : 1994년 1월 13일

I. 서 론

객체기반 프로그래밍 기법과 병행성을 결합하는 연구는 Actor모델에 의해 시작되었다. Actor모델은 객체연산의 장점과 함수연산의 장점을 결합한 것으로서 병행연산모델의 전형이 되고 있다.[1,2] 또한 많은 병행객체처리지향언어가 프로세스 개념의 CSP[3]와 CCS

[4]를 기반으로 개발되어 있다.[5] Actor모델은 함수연산의 특성에 의해 메시지의 비동기 전송과 단 방향 전송을 지원하며, 프로세스를 기반으로 하는 모델은 동기화된 메시지 전송과 양 방향 전송을 사용한다. Actor모델은 함수연산의 장점에 의해 병행성이 최대로 보장되며, 객체를 프로세스로 보는 관점에는 쓰레드(thread)에 의해 병행성의 정도를 결정하는 방법이 연구되고 있다.[3,4,5,6]

Actor모델에서의 객체는 상태변수를 갖지 않으므로 파이프라인 처리에 의한 병행성 증가에는 도움이 되지만 모델링의 개념이 어렵다.[7,8] 이 단점을 보완하기 위한 여러 연구 중 대표적인 ABCM/1[7,8]은 병행성을 최대한으로 보장해주지 못하며 메시지를 여러 가지 타입으로 정의하여 프로그래밍이 복잡하다. 또한 프로세스 기반 모델은 Actor모델 수준의 병행성을 지원하지 못한다.

두레모델은 위의 연구를 기반으로 하여 연산모델 개념의 단순화, 시스템에 의한 병행성의 검출, 최대의 병행성에 의한 연산 실행, 연산모델과 언어의 개념차이를 최소화하기 위해 연구되었다.

모델의 개념을 단순화하기 위하여 객체에 의한 캡슐화(encapsulation) 개념과 메시지 전송(message passing)에 의해 모델링과 연산의 진행을 실현할 수 있도록 하였다. 프로그램의 공유(sharing)와 재사용(reuse) 방법은 프로토타입(prototype)과 처리위임(delegation)의 개념을 사용한다. 프로토타입은 클래스(class)보다 모델링 면에서 더 좋은 방법으로 알려져 있으며[9,10], 처리위임은 상속(inheritance)에 비해 객체의 모듈성을 저해하지 않는 방법으로서 동적 재구성 면에서도 좋은 특성을 가지고 있다.[10,11,12]

병행성의 자동검출을 실현하기 위해서 태스크의 처리에 필요한 변수의 상관성을 검증할 수 있는 기준으로 MESet(Mutual Exclusion Set)을 정의하였다. MESet에 의해 동적 환경에서 생성되는 동적 객체는 Actor모델과 같은 병행성을 보장해 준다. 또한 객체의 연산 효율을 최대로 보장해 주기 위하여 연산을 기술한 메시지들을 병행으로 처리하도록 하였으며, 이 과정에서 변수의 상관성에 의한 문제를 해결하기 위해 Waiting Variable의 개념을 정의하였다.

두레모델에 의한 시스템을 기술하기 위해 개발된 두레언어 DL(Doorae Language)은 두레모델의 객체와 메시지 전송을 기술한다. 본 논문에서는 DL에 대한 자세한 설명은 제외하였으며 본 논문의 예제를 기술하는데 한하여 사용하였다.

본 논문의 II장에서는 두레 객체의 정의와 기본연산을, III장에서는 병행성의 한계와 자동검출을, IV장에서는 메시지 전송에 대하여 기술하고, V장에서 결론과 앞으로의 연구과제에 대해 기술한다.

II. 객체의 정의와 연산

두레의 객체는 모델링과 프로그래밍 단계에서 정의되는 객체와 동적 환경에서 생성되는 객체로 나누어진다. 정의되는 객체 중에서 대표적인 것은 모델링의 기본이 되는 프로토타입 객체이고, 동적으로 생성되는 객체 중에서 대표적인 것은 독립실행모듈(IPM : Independent Processing Module)이다. 프로토타입 객체와 IPM은 객체의 집합으로 다시 정의 된다. 두레의 모든 객체는 같은 구성을 가지고 있으며, 객체의 모든 연산은 세 가지로 정의된다.

1. 객체정의와 기본 연산

두레로 모델링 되는 시스템의 모든 요소는 객체로 정의된다. 두레의 객체는 자율적(autonomous), 동적(active)객체이며 객체의 집합에 의한 새로운 복합객체를 정의할 수 있다. 모든 객체는 시스템 내에서 유일한 메일 어드레스(mail address), 객체의 상태를 저장할 수 있는 변수(state variable), 메시지에 대한 연산(behavior)의 정의로 이루어진다. 이 때 메소드는 메시지 패턴의 집합으로 다시 정의된다. 모든 가능한 객체의 집합을 O, 모든 가능한 메일 어드레스의 집합을 A, 모든 가능한 상태변수의 유한부분집합의 모든 집합을 $F_S(S)$, 모든 가능한 연산의 집합을 B라 하면 두레의 객체는 정의 1과 같다.

정의 1. 객체: 객체는 메일 어드레스, 상태변수의 집합, 연산으로 이루어진다.

$$O = A \times F_s(S) \times B$$

두레시스템은 객체와 태스크에 의해 정의된다. 태스크는 처리가 완료되지 않은 메시지를 말하는데, 시스템 내에 태스크가 존재하지 않으면 시스템은 연산처리를 완료한 것이다. 태스크는 구분을 위하여 식별자를 가지며 메시지를 수신할 객체의 메일 어드레스와 메시지를 포함한다. 모든 가능한 태스크의 집합을 T, 모든 가능한 태스크의 식별자의 집합을 I, 모든 가능한 메시지의 집합을 M이라 하면 태스크는 정의 2와 같다.

정의 2. 태스크(task): 태스크는 식별자, 목적객체의 어드레스, 메시지로 구성된다.

$$T = I \times A \times M$$

객체는 태스크에 대해 연산을 행하는데, 객체의 기

본연산은 메시지 전송에 의한 새로운 태스크의 생성, 객체의 생성, 상태변수의 변경의 3가지이다. 메일 어드레스 a를 가지는 객체가 태스크에 대해 연산을 행한 결과는 정의 3과 같이 나타내진다.

정의 3. 객체의 연산: 객체의 연산은 태스크를 새로운 태스크의 생성, 객체의 생성, 객체의 상태변수의 변경으로 구성되는 집합으로 사상하는 과정으로 정의된다.

$$P=(I\times\{a\}\times M\rightarrow F_s(T)\times F_s(O)\times O)$$

정의 1, 2, 3에 의해, 메일 어드레스 a인 객체에서 태스크 (i,a,m)를 수행한 결과를 p(m)이라고 하면 연산의 결과는 p(m)=(T,O,o′)로 나타내진다. 이 때 T는 메시지 전송에 의해 생성된 태스크(메시지)의 집합이고, O는 새로 생성된 객체의 집합이며 o′는 상태변수가 변경된 객체 a를 나타낸다.

2. 프로토타입

프로토타입 객체는 복합객체로서 모델링의 기본이 되며 객체의 계속적(incremental) 정의를 지원하기 위한 방법으로 쓰인다. 두레에서의 프로토타입은 시스템 구성의 기본요소이며 시스템에 정의된 모든 프로토타입은 각각 독립적으로 연산을 수행할 수 있으므로 병행성의 기본이 된다. 프로토타입 객체의 DL에 의한 정의는 그림 1과 같다.[13]

```
(PROTOTYPE
    :NAME              prototype-name
    :ATTRIBUTES        static-states-list
    :PROTOCOLS
        (METHOD method-protocol-pattern-1
            :BEHAVIOR (
                    message-pattern-1
                    message-pattern-2
                    . . . ))
        (METHOD method-protocol-pattern-2
            . . . )))
```

그림 1. 프로토타입 객체의 정의

Fig. 1. Definition of Prototype object.

프로토타입은 실행과정에서 3가지의 모드를 가진다.[9,11,15] 프로토타입의 모드가 중요한 의미를 가지는 이유는 프로토타입이 정적상태변수(static state variables)를 가지고 있기 때문이다. 프로토타입이 태스크를 가지고 있지 않을 때를 DORMANT(휴면) 모드라 하는데, 이 때 외부로부터 메시지를 받으면 ACTIVE(활성)모드로 바뀐다. ACTIVE모드의 프로토타입이 변수의 값을 변경할 필요가 있을 때, 상호배제와 연산결과의 정확성을 보장하기 위해 WAITING(대기)모드로 바뀌며 이 모드에서는 다음 태스크에 대한 처리를 하지 않는다. WAITING모드에서 ACTIVE모드로 바뀌려면 변수 변경의 완료를 알리는 메시지를 받아야 한다.

태스크에 대한 프로토타입의 동작은 다음과 같다. 프로토타입은 모든 태스크에 대해 IPM(Independent Processing Module)을 생성한다. IPM은 프로토타입의 정적변수와 태스크의 처리 요구에 적합한 메소드로 구성된다. 프로토타입은 태스크에 의해 IPM을 생성한 뒤 해당 태스크를 생성된 IPM에 처리위임(delegate)함으로써 새로운 태스크를 생성한다. 생성된 IPM에 포함된 메소드가 프로토타입의 정적변수의 값을 변경하는 메시지 패턴을 가지고 있으면, 상호배제(mutual exclusion)를 실현하기 위해 IPM을 생성한 직후 프로토타입은 WAITING 모드로 바뀌며, 변수의 변경을 끝낸 IPM은 프로토타입에 완료 메시지를 보내어 ACTIVE모드로 바뀔 수 있도록 해야 한다.

3. IPM(Independent Processing Module)

두레시스템에서 병행연산을 수행하는 객체는 IPM이다. 프로세스의 관점에서 보면 IPM은 프로토타입 내부의 thread로 해석할 수 있으므로 두레의 프로토타입은 병행객체(concurrent object)이다.[5,6,10] IPM의 생성은 Actor모델에서 replacement연산을 하여 얻어지는 것과 같은 수준의 병행성을 제공하므로 시스템의 병행성은 시스템의 동작 초기에 정의된 프로토타입 개수만큼에서 IPM이 생성됨에 따라 동적으로 증가한다.

IPM의 생성은 DL을 비롯한 동적 환경에서 결정되므로 모델링과 프로그래밍을 쉽게 할 수 있다. IPM객체를 O_I라하고 프로토타입 객체를 O_P라 하면 O_I는 O_P의 부분집합 $O_I\subseteq O_P$로 정의된다. IPM생성은 프로토타입의 정적변수의 변경여부에 따라 나누어진다. 프로토타입의 상태변수의 집합 V중에서 어떤 태스크에 대한 연산과정에서 값이 변경되는 변수의 집합을 U (U ⊆ V)라하고, 그 값이 참조만 되는 변수의 집합을 R (R ⊆ V)이라 할 때 ($R\cap U=\phi$), $U=\phi$이면, 프로토타입은 이 태스크에 대해 IPM을 생성한 직후 다음 태스크를 처리하기 위한 IPM의 생성을 시작할 수 있다. 그러나 $U\neq\phi$이면 모든 변수 v (v ∈ U)는 상호배제(mutual exclusion)에 의해 보호되어야 하므로 프로토타입은 이 태스크에 대한 IPM을 생성한 뒤, 모든 v (v ∈ U)에 대한 변경이 끝나기를 기다리며 WAITING모드로 변경된다. 이

때, 이 IPM은 모든 v(v ∈ U)의 변경이 끝나면 WAITING모드의 프로토타입에 그 사실을 알려서 다음 태스크에 대한 연산을 시작할 수 있도록 해야 한다. 그림 2는 IPM의 생성과정을 보여준다.

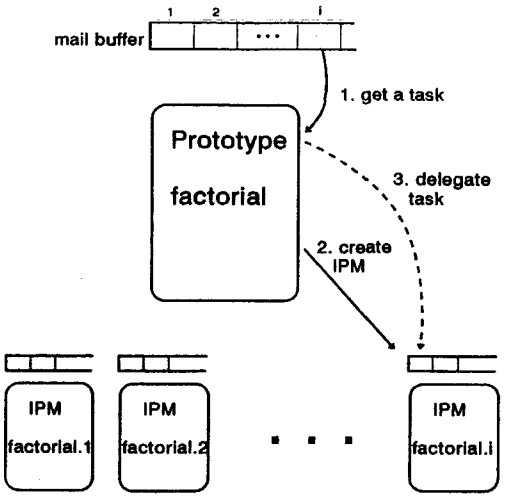


그림 2. IPM의 생성
Fig. 2. Creation of IPM.

Ⅲ. 두레의 병행성

두레모델에 의한 연산에서의 병행성은 여러 단계에서 검토해볼 수 있다. 프로그래밍에서 정의된 프로토타입 객체는 병행성의 가장 큰 단위이며 프로토타입은 자율적으로 동작하므로 시스템 동작 초기의 병행성 정도는 프로토타입의 개수이다. 프로토타입의 병행성은 연산을 진행하는 과정에서 IPM의 동적 생성과 소멸에 의해 변화된다. IPM의 생성에 의한 병행성은 연산의 실행에 관련하여 병행성의 기준이 되며 기존의 모델에서는 이러한 병행성의 검출을 프로그래머에게 맡기고 있다. 두레모델의 연구목적 중 하나가 되는 병행성의 자동 검출은 IPM의 생성규칙에 대한 것이며 이 규칙을 일반화하기 위해 GMESet-(Generalized Mutual Exclusion Set : 일반화된 상호 배제 집합)을 정의한다.

IPM이 연산을 진행하는 과정에서의 병행성은 메시지 전송의 독립성과 연산 의미(semantics)의 유지를 고려하여 최대한의 병행성을 유지할 수 있도록 하였다. 상호배제에 의해 연산의 의미를 유지하면서 이 목적을 달성하기 위해 WV(Waiting Variable)라는 개념을 정의한다. WV에 의한 병행성의 증대와 IPM 생성에 의한 병행성의 증대는 적용단계와 정도에 있어서 다른점이 있지만 의미론적으로는 동일하다.

1. 병행성의 자동검출

IPM단위의 병행성의 검출을 위한 기본 기준은 프로토타입의 정적상태변수의 변경여부에 있다. 커본적으로는, 어떤 프로토타입의 메일 버퍼에 수신된 태스크 $Task_1$, $Task_2$, . . .의 처리에 대한 변수참조(Read)와 변경(Update)에 사용되는 변수의 집합을 각각 $R_1, R_2, \cdots, U_1, U_2, \cdots$로 나타내면, $Task_1$에 대한 연산을 위해 IPM_1을 생성한 뒤, $U_1=\phi$일 경우에는 $Task_2$에 대해 IPM_2를 생성하여 병행처리를 실행할 수 있다. $U_1 \neq \phi$일 경우 IPM_1을 생성한 후 프로토타입이 block(WAITING 모드)되어 IPM_1의 연산이 끝난 뒤에 $Task_2$에 대해 IPM_2를 생성할 수 있다. 즉, 연산의 진행에 따라 시스템의 의미(semantics)를 유지하기 위한 상호배제(mutual exclusion)를 수행한 것이다.

이 관계를 보다 일반화하고 세분화하여 IPM간의 병행성을 자동으로 검출하고 가능한 최대의 병행성을 구현하기 위해 상호배제집합(MESet : Mutual Exclusion Set)의 개념을 정의한다. 위의 두 태스크에 대한 MESet이 $U_1 \cap (R_2 \cup U_2) = \phi$인 경우는 $Task_1$에 의한 변수 값의 변경이 $Task_2$의 병행 수행에 영향을 미치지 않으므로 $Task_1$과 $Task_2$를 병행처리할 수 있다. 이를 더욱 일반적으로 확장하면, 어떤 프로토타입에서 j번째 태스크에 대한 IPM의 생성을 결정할 때, 이전에 생성된 (j-1)개의 IPM중 현재 실행되고 있는 (j-i)개의 IPM($O_{Ii}, \cdots, O_{Ij-1}$)과의 관계에 따라 GMESet(Generalized MESet)을 정의할 수 있다. GMESet에 의한 j번째 IPM O_{Ij}의 생성 조건은 다음 정의 4와 같다.

정의 4. IPM생성 조건: 어떤 프로토타입의 j번째 IPM은 일반화된 MESet(Generalized MESet)에 대해 다음 조건이 만족할 때 생성가능하다.

$$GMESet \bigcup_{i \le k \le (j-1)} U_k \cap (R_j \cup U_j) = \phi$$

2. WV(Waiting Variable)에 의한 병행성의 증대

IPM의 연산 진행은 메시지에 의해 정의된 연산 코드를 순차적(sequential)으로 수행하는 과정이다. 이때 각 메시지는 연산 진행의 의미(semantics) 유지에 지장이 없으면 프로토타입의 IPM에 의한 병행성 구현과 마찬가지로 병행연산이 가능하다. 병행연산

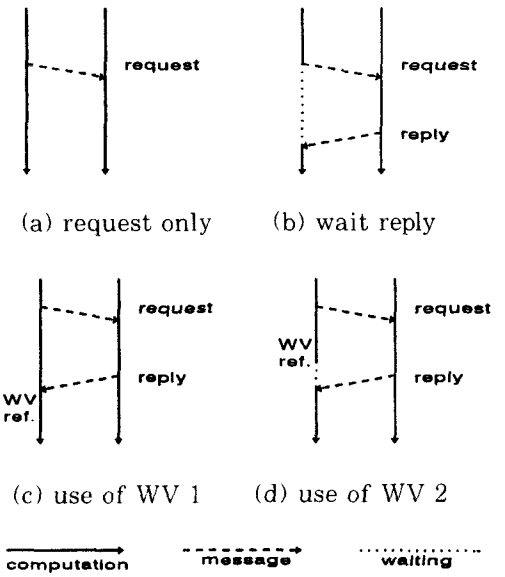
때문에 연산 진행의 의미가 영향을 받는 경우는 변수의 값을 치환하는 경우와 메시지에 대한 reply를 기다리는 경우이며, 이 때는 연산 진행의 의미를 유지하기 위해 병행성이 영향을 받을 수 있다.

```
(a)   (object_x :do_something)

(b)   (IF :(obj_X :do_something_and_reply)
          :THEN ( . . . )
          :ELSE ( . . . ))

(c)   (a := (object_X :do_something_and_reply))
```

그림 3. 메시지 형태의 종류
(a) request 메시지만 보내고 reply를 필요로 하지 않는 경우
(b) reply를 기다려야 하는 경우(조건문 등)
(c) reply를 기다리는 주체가 WV가 되는 경우

Fig. 3. Types of messages.

메시지에 대한 reply가 연산 진행의 흐름을 분기하기 위한 조건으로 쓰일 때는 reply에 의한 조건판단이 끝날 때까지 다음의 연산이 진행되지 못한다. 그러나 reply에 의해 변수 값을 변경하는 경우에는 그 변수의 참조여부에 따라 연산의 계속 진행이 결정된다. 이러한 경우에 프로시져를 기반으로 하는 기존의 언어들은 reply가 도착할 때까지 프로시져를 lock시키고, Actor는 reply를 필요로 하는 모든 경우에 대하여 customer라는 actor를 생성하여 reply를 기다리도록 한다. 그림 3은 몇 가지 메시지 유형에 대한 연산의 진행을 보인 것이다.



(a) request only (b) wait reply

(c) use of WV 1 (d) use of WV 2

그림 4. 메시지와 WV에 따른 연산의 진행

Fig. 4. Progress of computation according to messages and WV.

그림 4에서와 같이 실제로 연산을 수행하는 과정에서는 reply를 기다리는 모든 경우에 IPM이 block모드로 될 필요가 없다. reply에 의해 변수의 값을 변경하는 경우에도 값의 변경 후 그 변수가 최초로 참조되는 시점에 따라 연산의 진행이 결정된다. 즉, 연산의 진행은 reply에 의해 값이 변경되는 변수의 값이 참조되는 시점까지는 중단 없이 진행될 수 있다. ABCM/1에서는 메시지를 past, now, future의 세 가지로 나누고 각각에 대해 객체의 동작을 달리함으로써 이러한 문제를 해결하였다.[8]

AMCM/1의 해결책은 모든 경우에 병행성을 유지할 수 있도록 하지만 프로그래머의 판단과 프로그램에서의 명시적 표기를 전제로 한다. 두레에서는 이 부분에 있어서도 병행성을 자동으로 추출하고 암시적으로 지원하기 위하여 객체와 메시지에 의한 연산의 진행이라는 기본 전제를 적용하였다. 즉, 그림 3의 (c) 메시지에 의하면 변수 a는 객체 obj_X에서 보내올 reply를 받아 값을 치환하라는 메시지(:=)를 받는다. 이것을 변수도 객체라는 관점을 적용하여 해석하면 reply를 기다릴 책임은 변수 a가 속한 IPM에 있는 것이 아니라 변수 a에 있는 것으로 볼 수 있다. 따라서 변수 a는 WAITING모드로 상태를 바꾸고 reply를 기다리게 되고 IPM은 연산을 계속 진행해 나갈 수 있다. 변수 a가 reply값을 받으면 값을 치환하고 WAITING모드에서 벗어나게 되는데 이러한 과정은 IPM 연산의 흐름과는 독립적으로 수행된다(그림 4의 (c)). IPM의 계속 연산 진행 과정에서 WAITING모드에 있는 변수 a의 값을 참조하는 부분이 있는 경우에는 IPM의 연산은 진행을 중지하고 변수의 값이 치환되기를 기다려야 한다(그림 4의 (d)). 병행성의 증대를 위해 IPM의 연산 진행과 독립적으로 reply를 기다리는 변수를 WV(Waiting Variable)라 하고, WV의 검출은 프로그램의 컴파일시 행해져서 동적 환경에서 설정된다.

Ⅳ. 메시지 전송(Message Passing)

두레의 모든 연산은 객체에 대한 메시지 전송에 의해 이루어지는데, 메시지의 전송에는 몇 가지의 전제조건이 필요하다. 두레에서의 메시지는 일반적인 메

시지와 처리위임을 위한 메시지로 나누어 볼 수 있다. 두 가지 경우에 있어서 메시지의 전송형태는 같으나 연산의 진행에 대한 의미(semantics)에 있어서 차이가 있다.

1. 메시지의 정의

전송된 메시지는 반드시 원하는 객체에 도착되어야 하며, 전송된 메시지가 도착되지 않음으로 해서 생기는 모든 문제는 두레의 의미론에서 정의되지 않는다. 메시지는 점대점(point to point)방식으로 전송되며 방송(broadcasting)은 허용되지 않는다. 메시지를 전송하려는 객체는 메시지를 받을 객체의 이름(메일 어드레스)을 알아야 하며, 메시지의 전송은 비동기로 이루어지므로 모든 객체는 메시지를 저장할 버퍼를 가지고 있어야 한다. 또, 하나의 객체에서 다른 하나의 객체에 전송한 두 개의 메시지는 전송된 순서와 같은 순서로 도착해야 한다. 또한 메시지의 전송은 언제나 단 방향이다. 메시지에 의해 객체의 이름을 전송할 수도 있는데 새로 알게된 객체 사이에는 새로운 연결 통로가 생긴 것으로 볼 수 있다. 이 과정을 통하여 시스템의 구성(configuration, topology)이 변경되므로 동적 시스템 재구성(dynamic system reconfiguration)이 가능해진다.

메시지를 받은 객체는 자신이 가지고 있는 메소드 프로토콜 패턴과 같은 프로토콜을 가지고 있는 메시지에 대해 연산을 행할 수 있다. 프로토콜 패턴은 메소드의 이름과 인수의 리스트로 이루어지며 패턴의 동일성은 인수의 개수와 모든 인수의 이름을 검토하여 결정한다.

2. 처리위임(delegation)

처리위임은 계속적(incremental) 객체정의와 모델링을 지원하며, 코드의 재사용을 지원하기 위한 방법이다. 두레에서 사용하는 처리위임의 개념은 [10]에서와 같이 상속의 개념을 포함하는 일반적인 개념이다. 즉, 상속은 처리위임의 특별한 경우로 생각한다.[10,11] 상속에 대한 처리위임의 장점은 객체관계의 동적 특성과 초기 모델링의 편리성이다. 또한 정보와 프로그램 코드의 공유에 있어서도 우수한 특성을 보인다. 처리위임의 단점은 실행시 통신량이 증가하는 것이다.[11]

두레에서는 처리위임을 특정 태스크에 대해 행하기도 하고, 태스크를 받은 객체가 행할 수 없는 모든 태스크에 대해 행하기도 한다. 후자의 경우를 사용하면 상속을 실현할 수 있다. 그림 5는 처리위임에 대한 DL의 구문 예이다.

```
(METHOD (method-protocol-pattern @object-id))
        // 특정 메소드의 처리위임

(METHOD (any @object-id))
        // 모든 메소드의 처리위임, 상속의 기능 실현
```

그림 5. 처리위임의 구문
Fig. 5. Syntax of delegation.

V. 비교 및 고찰

두레의 모델링과 병행성의 평가를 위해 몇 가지 특징적인 면에서 Actor, ABCM/1과 서로 비교하였다. 연산모델의 비교에는 일반적으로 쓰이는 객관적인 기준이 없으므로 기존의 연구결과인 [16]을 인용하였으며, 두레모델과의 비교는 [16]을 인용하여 제시한 [15]를 재인용하였다. 이 비교표는 연산모델의 특성상 정량적 비교가 불가능하므로 정성적 비교방법을 이용하였다. 표 1은 연산모델의 특징으로서 현재 중요하게 생각되고 있는 것들을 비교한 것이다.[15,16]

표 1. ACTOR, ABCM/1, 두레의 비교
Table 1. Comparison of ACTOR, ABCM/1 and DOORAE.

	ACTOR	ABCM/1	두레
기본개념	함수연산과 객체 개념의 장점 결합	object-based 모델	object-based 모델
계산의 수행 방식	message passing과 delegation	좌동	좌동
분기 방식 (replacement)	사용자가 지정하는 explicit replacement, fine grain	explicit, method 단위의 분기만 보장, coarse grain	implicit replacement 보장, fine grain
병렬성 검출	프로그래머가 결정	좌동	context에 따라 시스템이 결정
병렬의 정도	fine grain	coarse grain	fine grain
인터럽트	queue만 사용하여 인터럽트 없음	express mode의 메시지 전송으로 인터럽트 처리	인터럽트 없음. 대기 상태에서 부분적 지원 가능.

두레의 모델링과 병행성의 실험을 위해 사용한 프로그램의 동작을 살펴보기로 한다. 예제로 선택한 문제는 IPM에 의한 병행성의 정도를 비교하기 위해 Actor에서 사용한 recursive factorial 문제를 선택하였고[1], WV에 의한 병행성의 증대를 보여주기 위해 조합문제를 추가하였다. 예제는 두레와 Actor, ABCM/1을 비교하기 위하여 각각의 구현언어로 쓰여진 프로그램을 구성하였다. 예제를 구현한 각 언어는 각각의 모델을 설명하기 위하여 쓰이는 언어이므로 모델들 간의 개념차이를 비교적 쉽게 알 수 있다.

조합 ${}_nC_r$의 풀이를 위한 프로그램은 그림 6과 같이 두 개의 프로토타입(actor, object)으로 정의할 수 있다. 각각의 프로그램에서 연산의 요청을 받은 뒤

연산과정에서 일어나는 병렬성의 정도는 아래의 예제에 대해서는 거의 동일하다. 아래의 예제에서 각각의

```
(PROTOTYPE
    :NAME       combination                 // 조합 nCr
    :PROTOCOLS
    (METHOD (do :n int :r int)
        :BEHAVIOR (
            (a := (factorial :do :n))
            (b := (factorial :do :r))
            (c := (factorial :do :n-r))
            (CF :printf :("the combination", n, "C", r, " is ..."))
            (CF :printf :(a/(b*c)))
            )))

(PROTOTYPE
    :NAME       factorial           // recursive factorial
    :PROTOCOLS
    (METHOD (do :n int)
        :BEHAVIOR (
            (IF :(n==0)
                :THEN (
                    ( 1 :reply_to :R))
                :ELSE (
                    ((n * (self :do :n-1)) :reply_to :R)
            ))
    ))
```

(a) DL

```
def exp Combination() (n, r)
    become Combination()
    let k = ((call Rec-Factorial(n)) * (call Rec-Factorial(r))
            * (call Rec-Factorial(n-r)))
    send k to output
end def

def exp Rec-Factorial() (n)
    become Rec-Factorial()
    if n=0
        then reply (1)
        else reply (n*(call self(n-1)))
    fi
end def
```

(b) Actor-SAL

```
[object combination
    (script
        (=> [:do-with n :and r]
            [temporary  [a := (make-future)]
                        [b := (make-future)]
                        [c := (make-future)]]
            [factorial <= [:do n) $ a]
            [factorial <= [:do n) $ b]
            [factorial <= [:do n) $ c]
            (if (and (ready? a) (and (ready? b) (ready? c)))
                [output <= (a * b * c)])))]

[object factorial
    (script
        (=> [:do n] where (= n 0)
            [R <= 1]
            (suicide))

        (=> [:do n] where (!= n 0)
            [R <= n * [(self-copy) <= [:do (n-1)]]]
            (suicide)))]
```

(c) ABCL/1

그림 6. $_nC_r$ 문제 풀이를 위한 예제 프로그램
Fig. 6. Example program for $_nC_r$.

경우에 차이가 나는 점은 병렬성의 구현에 있다. 예제 (a)의 두레의 경우는 병렬성을 표현하는 어떠한 구문도 포함되어 있지 않다. 예제 (b)의 Actor는 연산의 계속적인 수행과 파이프라인 방식의 병렬연산을 수행하기 위해 become 명령어를 수행시켜야 한다. 예제 (c)의 ABCL/1에서는 self-copy와 suicide명령에 의해 병렬성의 증가와 생성된 객체의 소멸을 프로그램에 표시하여야 한다. 그림 6과 같은 간단한 예제의 경우에는 세 가지 모델간의 큰 장단점을 발견하기 어렵다. 그러나 병렬성의 검출이 복잡하거나 어려운 경우, 또는 프로그램이 방대하거나 객체내의 지역변수가 중요한 역할을 하는 경우에는 프로그래머가 모든 경우에 일일이 병렬성의 검출과 기술을 하는 것은 상당히 힘드는 일이다. 위의 경우와 반대되는 경우로는, 프로그래머가 병렬처리에 경험과 지식이 많을 경우 일반화된 병렬화 기법으로 구현하지 못하는 훌륭한 프로그램을 작성할 수 있다. 따라서, 두레에서 채택한 방법과 기존의 방법이 각각의 장단점을 갖고 있다고 할 수 있으나, 현재와 같이 소프트웨어 공학의 개념이 도입되고 있는 현실에서는 두레의 방법이 더 유용하다고 생각된다.

그림 7은 그림 6 (a) 예제에서 두레모델의 연산 진행 과정에서의 IPM생성과 메시지 전송에 대한 그림이다. 프로토타입 combination은 태스크를 처리하기 위해 IPM combination.1을 생성한다. IPM combination.1은 조합을 계산하기 위해 세 개의 factorial계산을 프로토타입 factorial에 의뢰한다.

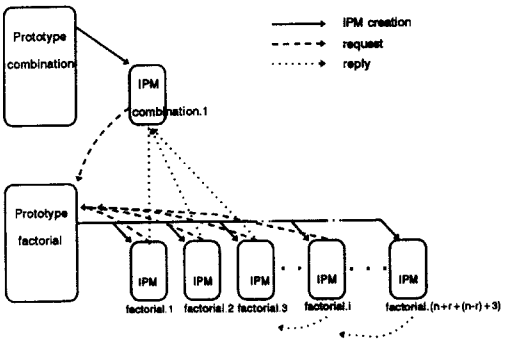


그림 7. $_nC_r$ 문제의 실행에 의한 IPM의 생성
Fig. 7. IPM creation in problem solving of $_nC_r$.

이 때 세 개의 계산은 모두 reply를 필요로 하며, 기존의 방법으로는 reply를 받기 위해 block되든지 ABCM/1과 같이 future 객체를 명시적으로 구분하

여 사용하여야 한다. 그러나 이 예제에서는 변수 a, b, c가 WV로 작용하므로 세 개의 factorial 계산을 병행으로 실행할 수 있다. 프로토타입 factorial은 세 개의 태스크에 대하여 세 개의 IPM factorial.i를 생성하고 각각의 처리를 맡긴다. 이 때 factorial은 궤환적으로 정의되어 있어서 생성된 IPM factorial.i는 인수의 값이 0이 될 때까지 프로토타입 factorial에 연산을 의뢰하므로, nCr을 계산하기 위해 프로토타입 factorial이 생성하는 총 IPM은 (n+r+(n-r)+3)개이다. 따라서 위의 계산은 총 (2n+4)개의 IPM에 의해 병행으로 연산이 실행됨을 알 수 있다.

(2n+4)개의 IPM이 시간의 변화에 따라 생성되고 소멸되는 과정을 그림 8에 나타내었다.

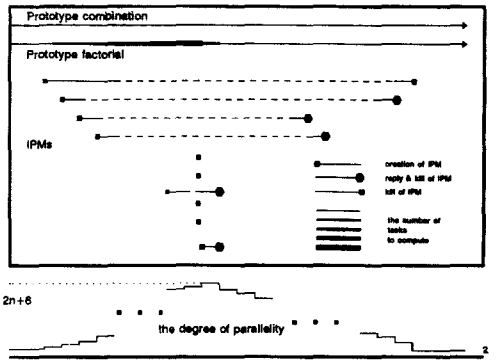


그림 8. 생성된 IPM의 실행시간과 시스템의 병행성
Fig. 8. Life times of created IPMs and degree of concurrency of system.

수평방향의 선들은 각 객체의 실행시간을 의미한다. 제일 위의 2개 직선은 프로토타입 combination과 factorial을 나타내며, 선의 굵기는 객체가 처리해야 할 태스크의 개수에 비례한다. 따라서 그림 8에 의하면 궤환적으로 정의된 factorial계산을 위해서 프로토타입 factorial에 태스크가 집중됨을 알 수 있다. 생성된 IPM들은 인수의 값이 0이 될 때까지 궤환 태스크를 생성하고, 그 태스크로부터 reply를 받기 위해 기다린다. 생성된 IPM중 가장 오래 실행되는 것은 combination.1으로서 세 개의 factorial계산이 모두 끝나기를 기다려 결과를 출력하고 소멸된다. 그림 8에서 아래 쪽의 그래프는 시간의 변화에 대한 시스템의 병행성의 변화를 보여준다. 시스템 전체의 최대 병행도는 두 개의 프로토타입을 포함하여 (2n+6)이다.

Ⅵ. 결 론

본 논문에서는 분산 및 병행 연산을 위한 연산모델 두레를 제안하였다. 두레모델은 모델링의 개념을 단순화하였으며, 병행프로그램의 구현시 장애가 되는 병행성의 검출을 자동으로 하게 하고, 연산수행시 동적 객체의 생성으로 최대의 병행성을 갖는 것을 목표로 하였다.

두레모델은 연산의 개념을 객체정의와 메시지 전송의 관계로 단순화함으로써 연산모델을 문제분석과 일치시키기 쉽게 하여 문제의 모델링을 쉽게 하였다. 또한 기존의 모델들이 병행성의 증대에 중점을 두어 프로그래밍 언어의 구현과정에서 연산모델과 많은 개념상의 차이를 가지던 점을 개선하여 DL에 의한 프로그램 작성시 연산모델의 개념을 그대로 사용하도록 하였다. 병행연산의 구현에서 병행성의 결정 과정과 프로그래밍 언어로 기술하는 과정의 복잡함을 줄일 수 있도록 병행성의 검출을 자동으로 할 수 있도록 하였다. 병행성의 판단과 검출을 지원하기 위해 정의한 GMESet은 동적 환경에서 병행처리의 가능성을 판정할 수 있는 기준을 제시해 준다. 또한 동적 환경에서 IPM을 생성하여 병행성의 증가가 동적 환경에서 실현되도록 하였다. 객체 내부의 병행성과 실행속도의 증가를 위해 정의한 Waiting Variable은 연산의 결과 값을 저장할 변수가 연산결과를 기다리도록 하여 IPM의 연산진행이 최대한 효율적으로 진행되도록 하였다. 예제 프로그램을 통하여 제안한 개념들과 병행성을 분석하였다.

현재 두레의 연산환경은 단일 프로세서 시스템에서 병행연산을 모의수행할 수 있도록 구현되어 있으며, DL의 컴파일러는 C언어로 작성되어 DL의 코드를 C코드로 변환하여 주도록 구현되었다. 현재 구현된 환경은 단일 프로세서 시스템에서의 모의수행이므로 실제 연산 속도는 C로 작성한 프로그램에 비해 모의수행의 부담 때문에 오히려 늦어진다. 그러나, 현재 연구하고 있는 워크스테이션 네트워크에서의 연산환경이 완성되면 속도는 워크스테이션 노드의 개수에 비례하여 증가하게 될 것이다. 또한, 두레의 연산을 위한 전용 컴퓨터 시스템을 개발하여 연산을 수행하면 그 효율은 더욱 높아질 것이다.

두레모델의 의미론적 정규화 모델에 대한 연구와 더불어 분산 병행 시스템에서의 연산실행을 위한 두레 컴파일러의 개선에 대한 연구를 진행중이다. 두레의 모델링 개념을 이용한 시스템 프로그래밍과 DB시스템, 지식기반 시뮬레이션[17] 등을 비롯한 분산 병행처리의 응용문제 해결과 두레시스템의 최적실행을 위한 컴퓨터 시스템의 개발에 대한 계속 연구가 필요하다.

參 考 文 獻

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Mass., 1986.

[2] C. E. Hewitt. "Viewing control structures as patterns of passing messages", *Journal of Artificial Intelligence* 8-3, pp.323-364, June 1977.

[3] C. Hoare. "Communicating Sequential Processes", *Comm. of ACM*, Vol. 21, No. 8, pp.666-677, Aug 1978.

[4] R. Milner. *Communication and Concurrency*. Prentice Hall International Series in Computer Science, 1989.

[5] M. Papathomas. "Concurrency Issues in Object-Oriented Programming Languages", in *Object Oriented Development*. Centre Universitaire d'Informatique, University of Geneva, ed. D. Tsichritzis, pp. 207-245, July 1989.

[6] M. Papathomas. *Language Design Rationale and Semantic Framework for Concurrent Object-Oriented Programming*. Ph.D. thesis, University of Geneva, 1992.

[7] A. Yonezawa, M. Tokoro editor. *Object-Oriented Concurrent Programming*. MIT Press, Cambridge Mass., 1987.

[8] A. Yonezawa. *ABCL: An Object-Oriented Concurrent System*. MIT Press, Cambridge Mass., 1990.

[9] H. Lieberman. "Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems", *Proc. OOPSLA '86*. pp.214-223, 1986.

[10] P. Wegner. "Dimensions of Object-Based Language Design", Proc. OOPSLA '87, pp.168-182, 1987.

[11] L. Stein. "Delegation is Inheritance", *Proc. OOPSLA '87*, pp.138-146, 1987.

[12] Won Kim, Frederic H. Lochovsky editor. *Object-Oriented Concepts, Databases, and Applications*. acm press, 1989.

[13] 김대권. *두래모델과 언어 사용자 시침서*. 연세대학교 전자공학과 내부문서, 1993

[14] Im Geun Lee, Ill Sang Choi, Dae Gwon Kim, Choong Shik Park, and Kyu Tae Park. "DOORAE : An Object-Oriented Concurrent Computational Model", *Proc. PRICAI 1992*, pp. 178-182, 1992.

[15] 이임건. *객체지향 병행계산모델 : 두래*. 연세대학교 전자공학과 석사학위 논문, 1993

[16] 한국 전자통신 연구소. *객체지향 언어의 구현을 위한 개발환경 구축에 관한 연구*. 1990

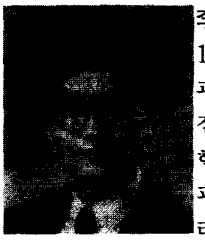
[17] Young Woon Woo, Min Suk Lee, Choong-shik Park, Jaihie Kim, In Pyo Hong, Moon Kim, Doong Lae Cho. "A Knowledge-based simulator Using the Prototype-delegation Method", *Proc. InfoScience "93*, pp.87-92, Oct 21-22, 1993.

著者紹介

金 大 權(正會員)

1988年 2月 연세대학교 전자공학과 학사. 1990年 2月 연세대학교 전자공학과 석사. 1993年 2月 연세대학교 전자공학과 박사수료. 1993年 3月 ~ 현재 연세대학교 공과 대학 산업기술 연구소 연구원. 주관심 분야는 병렬처리, 지식기반 시뮬레이션, 실시 간 시스템, 컴퓨터 구조등임.

•



李 林 建(正會員)

1991年 2月 연세대학교 전자공학과 학사. 1993年 2月 연세대학교 전자공학과 석사. 1993年 3月 ~ 현재 연세대학교 전자공학과 박사과정. 주관심 분야는 병렬 영상처리, 컴퓨터 구조, 영상 압축등임.

朴 忠 植(正會員) 第 30卷 B編 參照

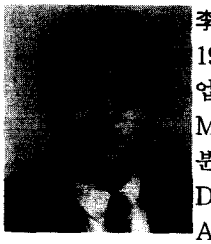
현재 永同 工科大學 전자계산학과 조교수

•

朴 圭 泰(正會員) 第 28卷 B編 11號 參照

현재 연세대학교 전자공학과 교수

•



李 溶 錫(正會員)

1973年 연세대학교 전기공학과 졸업. 1981年 University of Michigan, Ann Arbor, 반도체 분야 박사학위 수여. 그 후 Device Engineering, Gate Array, Memory, ISDN Telecommunication Chip, Microcontroller, RISC Microprocessor, Floating Point unit, cache Controller 등을 설계. IntelCorporation에서 Pentium 설계, 현재 연세대학교 전자 공학과 부교수