論 文
54D-8-3

페트리네트 기반 관리 제어시스템의 설계

Design of A Petrinet-based Supervisory Control System

孔 聖 學† · 徐 一 弘*
(Sung Hak Kong · Il Hong Suh)

Abstract - This paper presents a design experience of a supervisory control system. For effective programming of job commands, a petri net-type graphical language (PGL) is proposed applied to various tasks having concurrency and synchronization. Our PGL based supervisory control system is composed of PGL editor and PGL compiler; PGL editor is designed to help us to generate a job program using graphical symbols. PGL compiler includes analyzer, scheduler, and tranlator, PGL analyzer prevents a deadlock or resource allocation of unit cell, PGL scheduler generates a adequate job sequence of unit cell, and PGL translator translate the scheduled sequence into the job program of each unit cell.

Key Words : 페트리네트, PGL, 동기성, 동시성, 조건부 플레이스

1. 서 론

최근 수년간 자동화된 제조 분야에서 복잡하거나 정교한 작업을 수행하기 위해 작업 공간(work cell) 안에서 여러 대의 장치 모듈(device module)들을 제어할 수 있는 지능적인 제어기를 개발할 필요성이 증대되고 있다.[1] 새롭게 만들어지는 제어 시스템에는 복수 개의 장치 액추에이터(device actuator)와 센서들의 동시성과 실시간성을 고려하여 어떻게 작업 지시를 프로그래밍 할 수 있는가 하는 문제를 해결하여야 한다.

이를 위해 지금까지 대부분의 기존 제어 언어들이 특정한 동작들을 나타내는 의미 명령어로 구성된 작업 기술 형태로써 대부분 한 대 또는 두 대의 장치들만 제어하도록 개발되었다. 그러나 이러한 언어들은 충돌회피(collision avoidance), 태스크 계획(task planning), 다수의 제어기간의 통신(communication)등과 같은 기능적인 제약 때문에 광범위한 응용 면에서 한계가 있으며, 특히 그것의 내부적인 속성이 단위 동작들 간의 동기성(synchronization) 및 동시성(parallelism)을 표현하는데 적합하지 않다.[2][3][4][5] 이는 여러 개의 동작 모듈 및 다중 센서를 포함하는 작업 공간에서 수행할 작업을 여러 대의 제어기에서 프로그램 할 때 기존의 텍스트 기반(Text-based)언어로는 각 모듈들이 동시에 수행해야할 작업들 간의 동시성, 선행조건 등과 같은 상호관계를 표현할 수 없거나 있다 하여도 초보적인 단계에 머물러 사용함에 불편함이 있었다. 특히 여러 개의 단위 모듈로 구성된 복잡한 시스템에서 각각의 단위 모듈에 수행해야 할 단위 작업의 흐름을 지시하기 위해서 대부분의 제어기에서는 텍스트로 표현된 작업 명령을 작업 순서에 따라 맞게 나열함으로써 각 모듈에 작업을 지시하였다. 이러한 작업 지시 방법은 한 대의 제어기에 대한 작업을 프로그램 할 경우 실제 모듈 작업의 흐름과 텍스트로 표현된 한 대의 모듈에 대한 프로그램의 흐름이 동일하므로 매우 유용하게 사용되며 이해하기에 편리하다는 장점이 있다. 그러나 이러한 텍스트 기반 제어 언어를 이용하여 두 대 이상의 모듈에서 동작하는 작업을 기술한 경우 제어기의 프로그램의 흐름은 하나인데 반해 실제 작업의 흐름은 두 가지 이상이 되므로 사용하기 어려웠다. 이를 해결하기 위한 연구들을 보면, 기존 텍스트 기반 제어언어에 두 대의 로봇을 위한 동시 작업 명령어들을 추가한 제어언어들이 제안되었고[14], 여러 단위 모듈간의 관계를 표현하기 R. Pattipati et. al.[6]는 마코비안 모델(markovian model)을 이용한 방법을 제안하였고, Brandin[7]은 큐잉 네트워크(queueing network)와 유한 상태 머신(finite state machine)을 이용한 방법을 제안하였고, M. S. Kim et. al.[8]은 논리적인 모델(logical model)을 이용한 방법을 제안하였다. 그러나 이러한 연구들은 주어진 시스템들을 해석하거나 평가하는 방법에 관한 연구들이었지 작업을 지시하기 위한 제어 언어에 관한 연구는 아니었다. 또한 I. H. Suh et. al.[9]는 페트리네트를 이용해 태스크간의 작업을 생성하는 방법을 연구하였으나, 실행될 태스크간의 경로 선택 문제를 일으킬 수 있다는 단점이 존재한다.

그래서 본 논문에서는 여러 작업을 동시에 제어하기 위한 제어 언어로써 페트리네트를 이용한 새로운 제어 언어를 제안한다. 원래 페트리네트는 기계, 전자, 전기, 컴퓨터 등 여러 분야에서 주어진 시스템을 모델링하여 해석하거나 설계하는 수단으로 유용하게 사용하는 방법이다. 로봇 분야에서

† 교신저자, 正會員 : 漢陽大學校 電子通信電波工學科 博士課程
E-mail : shkong@incorl.hanyang.ac.kr
* 正 會 員 : 漢陽大學校 情報通信大學院 教授 · 工博
接受日字 : 2005年 4月 25日
最終完了 : 2005年 6月 15日

도 로봇이 사용되는 작업 공정을 페트리네트로 모델링하여 공정 설계의 최적화 및 작업 공정의 생산성 등 성능을 평가, 분석하는 수단으로 많이 이용하고 있다[10][11][12]. 제안된 제어 언어에서는 주어진 시스템 또는 작업을 모델링하고 평가, 해석하는 기존의 응용방식과는 달리 그래픽 작업 편집기 상에서 페트리네트를 이용하여 제어기가 수행해야할 작업을 프로그램 작업 기술 수단으로 페트리네트를 이용하였다. 각 모듈의 작업을 기술하는데 있어서 동시성의 표현이 뛰어난 페트리네트를 이용함으로써 전체 시스템의 작업을 효율적으로 프로그램 할 수 있도록 하며, 페트리네트로 기술된 작업 프로그램을 해석하여 각 단위 모듈이 수행 할 작업들 간의 표착상태(deadlock) 및 자원공유문제(resource allocation) 등을 사전에 검색하여 논리 오류(logical error)를 방지할 수 있게 하였다. 또한 작업간의 충돌시 주어진 조건에 따라 다음 작업을 선택하도록 하는 방법으로 조건부 플레이스(conditional place)를 추가적으로 제안한다.

본 논문의 구성은 다음과 같다. 2절에서는 페트리네트에 관한 정의를 설명하였고, 3절에서는 제안한 시스템의 PGL의 구현 및 해석 방법들에 관하여 기술하였으며, 4절에서는 제안한 시스템의 PGL를 이용한 작업 프로그램 예제들에 관하여 기술하였으며, 5절에서는 결론 및 추후 과제를 기술한다.

2. 페트리네트

페트리네트는 1962년에 C. A. Petri에 의해 제안되었다. 페트리네트의 장점은 동시성(concurrency)과 동기적인 사건(synchronized event)을 표현할 수 있으며, 이를 가시적으로 표현 가능하며 이해하기 편리하다는 장점이 있다[16][17].

페트리네트는 일반적으로 사건은 트랜지션(transition)으로 표현한다. 트랜지션이 발생하기 위해서는 시스템의 여러 조건이 만족되어야 한다. 이러한 조건에 대한 정보는 플레이스(place)에 저장된다. 어떠한 플레이스들은 트랜지션의 입력 조건이 되고, 어떠한 플레이스들은 출력 조건이 된다. 이러한 관계를 통하여 페트리네트 다음과 같이 정의된다.

정의 1. 페트리네트는 아래의 4-튜플(4-tuple)로 구성된다.

$$(P, T, A_i, A_o) \tag{1}$$

여기서 P는 플레이스 집합(place set)으로 $P=\{p_1, p_2, p_3, \cdots\}$로 표현하고, T는 트랜지션 집합(transition set)으로 $T=\{t_1, t_2, t_3, \cdots\}$로 표현한다. 아크 함수 A_i는 플레이스로 들어오는 트랜지션의 연결정보를 나타내며 입력 따름 행렬(input incidence matrix)라 하고 $\alpha(p,t)$로 표기한다. 아크 함수 A_o는 플레이스에서 나가는 연결정보를 나타내며 출력 따름 행렬(output incidence matrix)라 하고 $\alpha(t,p)$로 표기한다.

페트리네트는 이산현상시스템(discrete event system)의 동작을 표현하기 위해 토큰(token)의 개념을 도입한다. 토큰은 각 플레이스에 부여된 상태값이다. 앞에서 기술한 것 같이 플레이스는 시스템의 이벤트를 발생시키기 위한 조건정보를 담고 있고, 그 조건정보가 토큰의 개수로 표현된다. 그러므로 각 플레이스에 부여된 토큰의 개수들의 정보로 시스템의 상태를 나타낸다.

정의2.페트리네트의마킹 m이란아래의함수와같다.

$$m: P \rightarrow \{0, 1, 2, \cdots\} \tag{2}$$

따라서 마킹 $m=\{m(p_1), m(p_2), \cdots, m(p_n)\}$이고, n은 페트리네트 내부의 플레이스의 개수이다.

정의3.표기페트리네트(Markedpetrinet)

표기 페트리네트 아래의 5-튜플로 구성된다.

$$(P, T, A_i, A_o, m_0) \tag{3}$$

여기서 (P, T, A_i, A_o)는 페트리네트이고, m_0는 초기 마킹이다. 표기 페트리네트는 초기 마킹을 기반으로 시스템의 동작을 표현된다. 페트리네트에서 마킹은 이산 현상시스템의 상태를 나타낸다. 상태와 상태의 천이는 인에이블 규칙과 점화 규칙에 의해 따라 동작한다.

정의4.인에이블규칙(Enablerule)

어느 트랜지션의 입력 플레이스에 대하여 아래의 조건을 만족하면 그 트랜지션은 인에이블 되었다고 한다.

$$\forall p \in \cdot t, a(p,t) \le m(p) \tag{4}$$

인에이블된 트랜지션들은 점화가 가능하며 트랜지션의 점화에 의해 표기 페트리네트의 마킹은 점화 규칙에 의해 변화한다.

정의5.점화규칙(Firingrule)

트랜지션 t가 점화될 때 현재의 마킹 m은 새로운 마킹 m'으로 아래와 같이 변화 한다.

$$\begin{aligned} &\forall p \in \cdot t, m'(p) = m(p) - a(p,t) \\ &\forall p \in t \cdot, m'(p) = m(p) + a(p,t) \end{aligned} \tag{5}$$

페트리네트의 점화규칙에 따라 트랜지션 t가 점화될 때 현재의 마킹 m은 새로운 마킹 m'으로 아래와 같이 변화 한다.

$$\begin{aligned} &\forall p \in \cdot t, m'(p) = m(p) - a(p,t) \\ &\forall p \in t \cdot, m'(p) = m(p) + a(t,p) \end{aligned} \tag{6}$$

앞서 m_0에서 $\sigma = t_1 t_2 t_3 t_4 \cdots$의 트랜지션에 의해 m'로 도달하는 것을 $m[\sigma\rangle m'$로 나타낼 수 있고, 이에 대응하는 점화 순서는 $u_1 u_2 u_3 u_4 \cdots$로 나타낸다. 여기서 u_k는 t_k의 점화를 벡터형태로 나타낸 것으로 하나의 원소에서만 1의 값을 가지고 나머지는 0인 열벡터이다. 이를 상태 방정식으로 표현하면 아래와 같다.

$$m_k = m_{k-1} + (A_i - A_o) u_k, \; k = 1, 2, \cdots \tag{7}$$

여기서, A_i는 입력 따름 행렬, A_o는 출력 따름 행렬 그리고 $A = A_i - A_o$는 따름 행렬(incidence matrix)라고 한다. 그리고 초기 마킹에 대하여 표현하면,

$$m_k = m_o + A \sum_{i=1}^{k} u_i \tag{8}$$

이다. 각 플레이스의 토큰 정보는 마킹 벡터로 표현된다. 그리고 트랜지션의 점화 역시 벡터 형식 나타낼 수 있는데 이를 점화 벡터(firing vector)라고 한다. 점화 벡터는 $x \in Z^{|T| \times 1}$의 정수로 이루어진 벡터이다. 각 i번째 원소는 $t_i (i = 1, 2, \cdots, |T|)$의 점화 횟수를 나타낸다. 그러면, 점화 벡터 x에 의해 마킹의 변화는 아래의 상태 방정식으로 쓸 수 있다.

$$m' = m + A_i \cdot x - A_o \cdot x$$
$$= m + A \cdot x \quad (9)$$

위의 식을 변형하면,

$$Ax = \triangle m \quad (10)$$

여기서, $x = \sum_{k=1}^{d} u_k$이고 $\triangle m = m_d - m_0$이다.

또한, 인에이블 조건에 의해 x가 점화가능하기 위해서는 아래를 만족해야 한다.

$$m_0 - A_i \cdot x \geq 0 \quad (11)$$

따라서 도달가능성 문제는 아래와 같이 나타낼 수 있다.

정리. 페트리네트 (N, m_0)으로 주어질 때, m'는 m_0로부터 도달가능이면 $Ax = m' - m$ 이고 $m_0 - A_i \cdot x \geq 0$인 음수가 아닌 정수로 이루어진 열벡터 x가 존재한다. 그 때의 열벡터 x는 m_0 에서 m'으로 갈 때의 점화벡터이다.

3. 제안된 Petrinet-type Graphical Language 시스템

본 논문에서는 여러 작업을 동시에 제어하기 위한 제어 시스템 언어로써 페트리네트를 이용한 새로운 제어 언어를 구현하였다.

구현된 PGL은 그림 1과 같은 구조로 되어 있으며, 전체적인 구성은 다음과 같다.

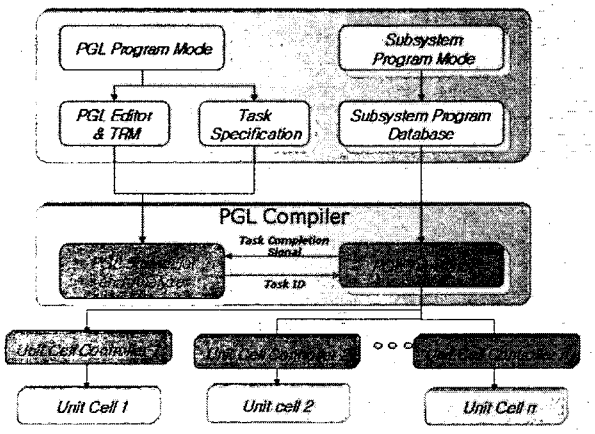


그림 1 PGL 유닛 구조
Fig. 1 PGL unit architecture

- 그래픽 편집기상에서 플레이스, 트랜지션, 아크와 같은 그래픽 심볼(graphical symbol)을 이용하여 작업 지시 프로그램을 작성하는 PGL 편집기

- 페트리네트로 기술된 작업 프로그램을 해석하여 각 유닛들이 수행할 작업들 간의 교착상태(Deadlock) 및 자원 공유 문제(Resource Allocation) 등을 사전에 검색하여 논리오류를 방지할 수 있도록 도와주는 PGL 분석기(analyzer)와 각 유닛에 적합한 작업 교시 방법으로 변환하여 주는 PGL 변환기(translator)로 구성된 PGL 컴파일러(compiler)

3.1 PGL 편집기

페트리네트를 이용하여 시스템을 표현하면 사건과 조건의 두 가지 기본적인 개념으로 축약된다. 사건은 시스템에서 일어나는 동작(action)을 뜻하며, 사건의 발생은 시스템의 상태에 따라 결정된다. 또한 시스템의 상태는 조건으로 표현되며, 참 또는 거짓의 값을 가진다. 따라서 시스템의 조건을 페트리네트의 플레이스로 표현하고, 사건을 트랜지션으로 표현하여 시스템의 특성을 쉽게 모델링할 수 있다[12]. 본 논문에서는 페트리네트를 이용하여 전체시스템의 작업 흐름을 프로그램 하였다. 전체 시스템의 각 단위 태스크를 PGL의 각 플레이스에 할당하며, 각 플레이스는 전체 시스템이 해당 모듈에 태스크를 수행 중인 상태를 나타내고 태스크가 완료되면 플레이스의 조건이 만족된다.

트랜지션은 출력 플레이스가 의미하는 태스크의 수행 시작을 나타내는 사건을 말하며, 사건이 발생하기 위해서는 해당 트랜지션의 선행 조건이 만족되어야 한다. 즉, 입력 플레이스가 의미하는 태스크의 수행이 모두 완료된 상태이어야 트랜지션이 인에이블 상태가 된다. 이러한 PGL을 이용하여 사용자가 전체 시스템의 작업을 프로그램 할 수 있도록 본 논문에서는 그림 2와 같은 윈도우즈(windows)를 이용한 그래픽 편집기(graphic editor)를 구현하였다.

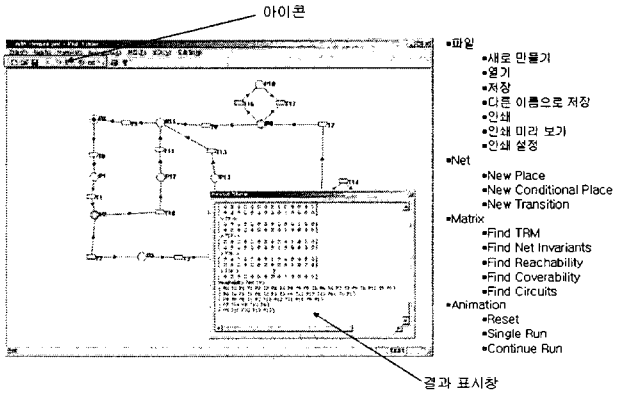


그림 2 PGL 편집기
Fig. 2 PGL Editor

3.2 제안된 PGL 방법의 해석 알고리즘

본 논문에서는 페트리네트를 이용해 시스템 혹은 작업을 모델링하고 이를 평가, 해석하는 기존의 응용 방식과는 달리 여러 대의 장치 모듈로 구성된 복잡한 시스템의 작업을 교시하는 제어 언어로 구현하였다. 이렇게 페트리네트를 이용하여 작업을 표현 또는 기술하는데 있어 고려해야 할 사항으로써 충돌(conflict) 또는 혼동(confusion) 문제가 발생된다. 이러한 문제들은 공유할 수 없는 동일한 자원을 사용하는 두 개 이상의 트랜지션이 동시에 수행될 때 발생한다. 이러한 문제는 페트리네트를 이용해 시스템을 평가, 해석하는 기존의 방법에서는 크게 문제되는 것은 아니다. 그러나 본 연구에서와 같이 해석된 결과를 이용해 작업을 지시하는 작업 파일을 생성하는 경우에는 문제가 발생된다. 즉 자원 공유에 따른 자원 우선 할당 문제, 작업된 상황에 따라 작성

되는 작업 순서 문제를 표현할 수 없는 문제가 발생된다. 그래서 본 논문에서는 페트리네트에 관한 정의를 확장해 조건부 플레이스(conditional place)라는 것을 추가적으로 정의하여 이 문제들을 해결하였다.

3.2.1 조건부 플레이스

그림 3과 같이 p1에 토큰이 다음 트랜지션인 t2와 t3 중 어떤 트랜지션에 할당되어하는지를 표현하기 어렵다, 이러한 문제를 기존의 페트리네트에서는 금지 아크 페트리네트(inhibit arc petri nets)나 우선순위 페트리네트(priority petri nets)를 이용해 표현하였다[18]. 그러나 금지 아크 페트리네트인 경우는 할당해야하는 트랜지션의 수에 따라 정의해야하는 상태의 수가 2^n 만큼씩 증가하는 복잡성의 문제가 있고, 우선순위 페트리네트인 경우는 처리하는 태스크의 우선순위가 고정된다는 단점이 있다.

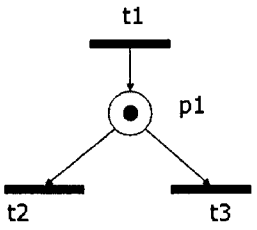


그림 3 충돌 문제
Fig. 3 Conflict problem

그래서 본 논문에서는 이러한 문제들을 해결하기위해 조건부 플레이스라는 것을 정의하였다. 조건부 플레이스는 일반적인 플레이스에 다음 트랜지션이 토큰이 전이되는 조건 특성을 추가할 수 있도록 하였다. 즉 조건부 플레이스는 주어진 조건에 따라 다음 트랜지션이 토큰이 전이되어 점화시킨다. 그림 4는 제안된 조건부 플레이스이다. 조건부 플레이스의 동작은 조건부 플레이스에서는 p1에 할당되는 조건에 따라 t2 또는 t3 중 한 개의 트랜지션만을 점화 시키도록 구현하였다.

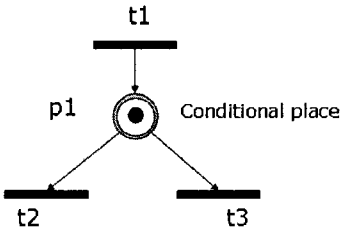


그림 4 조건부 플레이스
Fig. 4 Conditional place

그렇게 함으로써 해석적으로는 가존에 페트리네트 해석방법들을 장점을 이용하면서도 보다 직관적인 표현이 용이하도록 하였다. 기능적으로는 PGL 프로그램 내에서 자원 공유인 경우에는 세마포와 같은 기능, 경로 선택 문제인 경우는 플래그와 같은 기능을 수행하도록 하였다.

3.2.2 자원 공유 문제

자원 할당 문제란 공유할 수 없는 동일한 자원을 사용하는 두 개 이상의 태스크가 동시에 수행될 때 발생하며, 이때 서로 사용하고자 하는 자원을 어떤 태스크에게 할애할 것인가를 결정하는 문제를 말한다. 예를 들어 그림 5와 같은 PGL 프로그램에서 태스크 A와 태스크 B가 동일한 비젼(vision)을 사용하는 경우 이 두 개의 태스크가 동시에 수행될 때 충돌이 발생하게 된다. 이를 위해 기존 방법에서는 태스크 A 와 태스크 B에 고정된 우선순위를 부여하고 부여된 우선순위에 따라 태스크가 처리되는 방법을 사용하였다. 그러나 이러한 방법은 항상 고정된 우선순위에 따라 태스크를 수행하게 되므로 수동적인 방법이라고 할 수 있다. 그래서 본 논문에서는 이러한 문제를 보다 능동적으로 해결하기 위하여 세마포(semaphore) 역할을 담당하는 조건부 플레이스를 이용하여 충돌이 예상되는 태스크간의 상호배반(mutual exclusion) 표현을 가능하도록 구현하였다. 즉 그림 5와 같이 충돌이 예상되는 두 태스크를 실행시키는 각각의 트랜지션이 선행 조건으로 세마포를 갖도록 하여 먼저 작업을 시작하는 태스크가 작업을 마칠 때까지 다른 태스크의 수행을 금지시킴으로써 충돌을 예방하였다.

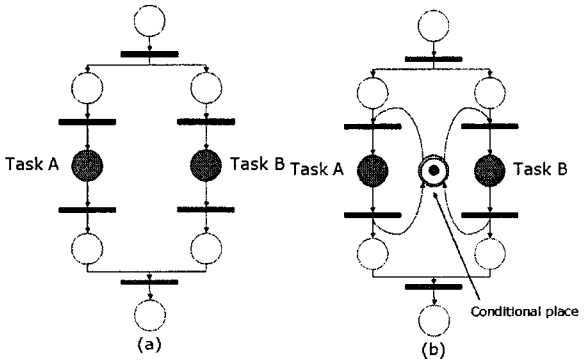


그림 5 동일한 자원을 공유하는 두 개의 태스크
Fig 5. Two task with same resource sharing

3.2.3 작업 경로 선택의 문제

페트리네트를 이용해 작업을 표현하는데 또 따른 문제가 발생한다. 한 개의 플레이스에서 두개 이상의 트랜지션을 동시에 수행될 때 발생하며, 그림 6에서와 같이 p1에서 토큰이 t2 t3 중 어떤 트랜지션에 할당해 주어야 하는가를 결정해야한다. 예를 들이 p1에서 부품 검사를 하여 합격이면 경로 1(path 1)을 따라 작업하고, 불합격이면 경로 2(path 2)를 따라 작업하고자 경우이다. 이것을 표현하기 위해 기존에서는 금지 아크 페트리네트를 사용하였다. 금지 아크란 플레이스에서 트랜지션으로 연결되는 방향성 아크를 두어 해당 플레이스에 토큰이 존재하면 해당 트랜지션의 점화를 금지시키는 방법이다. 이러한 방법의 단점은 할당해야하는 트랜지션의 수에 따라 각각의 상태를 정의하고 이를 논리적으로 표현해야 한다. 따라서 정의하는 상태의 수는 2^n가지나 된다. 여기서 n은 할당할 트랜지션의 개수이다. 그래서 본 논문에서는 조건부 플레이스의 속성에 플래그와 같은 역할을 담당하도록 구현하였다. 구현된 조건부 플레이스에 따라 p1에서 부품 검사 결과라는 플래그를 설정하면 설정된

플래그에 따라 t2 또는 t3에 토큰이 전달되어 다음 작업을 수행하는 작업 경로를 수행하도록 구현 하였다.

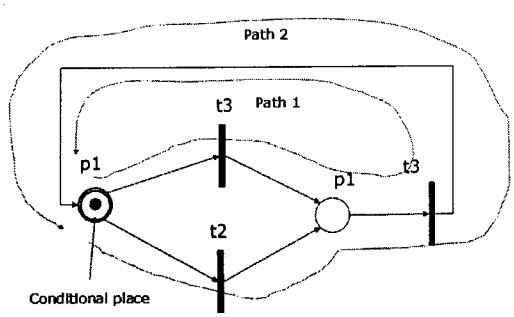


그림 6 경로 선택의 예
Fig. 6 Example of path selection

3.2.4 PGL의 TRM 생성

PGL에서 플레이스 집합을 $P=\{p_1, p_2, p_3, \cdots, p_n\}$, 트랜지션 집합을 $T=\{t_1, t_2, t_3, \cdots, t_n\}$로 표현하고 플레이스와 트랜지션 사이의 아크를 (p_i, t_j)또는 (t_j, p_i)로 표현할 수 있다. PGL 프로그램을 기술하는 더욱 편리한 표현으로 트랜지션에 대한 입력 플레이스 집합을 $I(t_j)$로 표현하고, t_j 대한 출력 플레이스 집합을 $O(t_j)$로 표현할 수 있으며, A는 아크의 집합을 나타낸다.

$$I(t_j)=\{p_i:(p_i,t_j)\in A\}, O(t_j)=\{p_i:(t_j,p_i)\in A\} \quad (12)$$

플레이스 p_i에 대해서도 $I(p_i)$와 $O(p_i)$의 표현이 적용되며, 플레이스 p_i로부터 트랜지션 t_j로 향한 아크는 $p_i\in I(p_j)$ 및 $p_j\in O(p_i)$를 의미한다.

편집기에서 PGL 프로그램이 작성되면 PGL 컴파일러는 아크에 의한 연결 관계를 인식하여 m개의 플레이스와 n개의 트랜지션 간의 관계정보행렬(Task Relation Matrix, TRM)을 생성한다. TRM에서 i번째 행은 p_i에 대한 트랜지션과의 관계를 나타내고 j번째 열은 t_j 에 대한 플레이스 관계를 나타낸다. 여기서 1은 $(p_i, t_j)\in A$ 즉, p_i 에서 t_j로 아크가 있는 것을 의미하고, -1은 $(t_j, p_i)\in A$ 즉, t_j 에서 p_i 로 arc가 있는 것을 의미하며, 0은 p_i 와 t_j 사이에 아크가 없는 것을 의미한다.

3.2.5 PGL의 해석

본 논문에서는 페트리네트 해석에 있어서 고려되는 여러 가지 성질 중 제어 시스템 작업 해석에 필요한 safeness, liveness, 교착상태의 해석에 대하여 연구하였다. safeness는 페트리네트를 이용하여 하드웨어 장치(hardware device)를 모델링할 경우 고려해야 할 성질로써 플레이스의 토큰의 수가 1을 넘지 않을 때 safe하다고 한다. 이러한 검사를 통해 시스템의 해당 플레이스가 작업을 수행 중 다시 실행 명령이 내려지는 오류를 예측할 수 있다. 또한 liveness와 교착상태는 컴퓨터 시스템의 자원 할당에 있어서 제기되는 문제로 liveness는 트랜지션이 점화될 수 있는지 여부를 검사하는 것이며, 교착상태는 동시에 수행되는 두 개 이상의 태스크가 서로 상대방이 사용하고 있는 자원을 기다리고 있어 작업이 더 이상 진행될 수 없는 교착상태를 말한다.

페트리네트를 해석하는데 사용하는 일반적인 방법으로는 특정 페트리네트에 대한 도달 가능 트리(reachability tree)를 이용하는 방법이 있다. 도달 가능 트리를 이용하는 방법은 페트리네트의 초기 마킹 상태로부터 가능한 모든 마킹 상태를 그려 나감으로써 트리를 구성하는 것이다. 이렇게 작성된 트리에서는 가능한 모든 마킹 상태와 이에 이르기 위해 필요한 트랜지션이 점화되는 경로가 나타나므로 위에서 언급한 세 가지 문제점이 일어나는지 여부를 알 수 있다. 즉 트리에 나타난 모든 마킹 중에 플레이스가 2개 이상의 토큰을 갖는 경우가 발생하는지를 검사하여 safeness를 알 수 있으며, 트랜지션이 점화되는 경로가 한번도 나타나지 않는 트랜지션은 live하지 않음을 알 수 있다. 또한 트리의 구조가 특정 마킹 상태에서 더 이상 진행하지 못하게 되면 그 부분에서 교착상태가 발생할 수 있음을 예측할 수 있다. 예를 들어 그림 7과 같은 두 개의 프로세서로 구성되고 P3과 P4를 상호 교차되어 공유함으로써 교착이 예상되는 페트리네트에서 초기 마킹 상태 $M(0)$는 $[1,0,0,1,1,1,0,0]$과 같다. 여기서 [ ]안의 요소들은 순서대로 P0, P1, P2, P3, P4, P5, P6, P7이 초기에 갖는 토큰의 수를 나타낸다. 이러한 초기 상태에서 인에이블되는 트랜지션 T8과 T11이 점화되었을 경우의 마킹을 구해 나가는 방법으로 트리를 구성하면 그림 8과 같다.

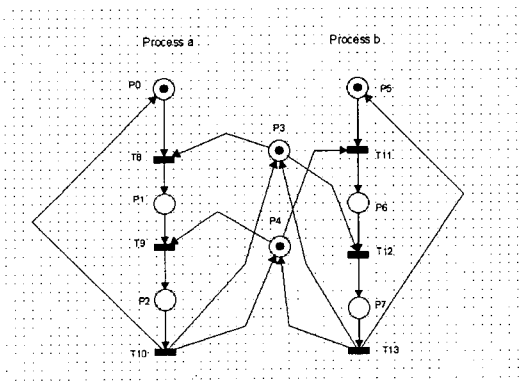


그림 7 두 개의 태스크가 수행되는 작업
Fig. 7 Two task complemented simultaneous

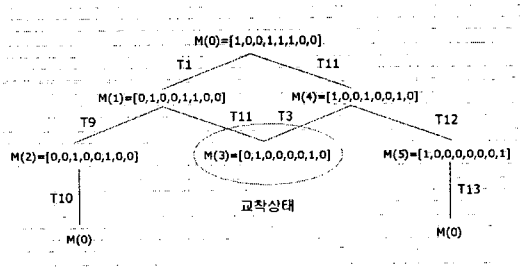


그림 8 그림 7에 대한 도착 가능 트리
Fig. 8 Reachability tree for Fig. 7

위와 같은 트리에서 각 마킹을 연결하는 화살표에 표시된 트랜지션은 화살표가 가리키는 플레이스에도달하기 위해 점

화되어야 하는 트랜지션을 나타낸다. 이와 같이 작성된 트리에서 M(2) 및 M(5)에서 각각 T10과 T13이 점화 되면 초기 마킹 상태 M(0)로 돌아오지만 M(3)에서는 더 이상 인에이블된 트랜지션이 없는 교착상태가 발생됨을 알 수 있다.

트리의 자동 생성 방법은 초기 마킹 상태인 트리의 루트(root)로부터 시작해서 왼쪽에서 오른쪽의 순서로 각 노드(node)의 자식 노드(child node)를 재귀적(recursively)으로 생성해 가는 깊이 우선 탐색 알고리즘(depth-first traversal algorithm)을 이용하여 트리를 생성하게 되며, 이러한 PGL 분석기의 알고리즘 흐름도는 그림 9와 같다.

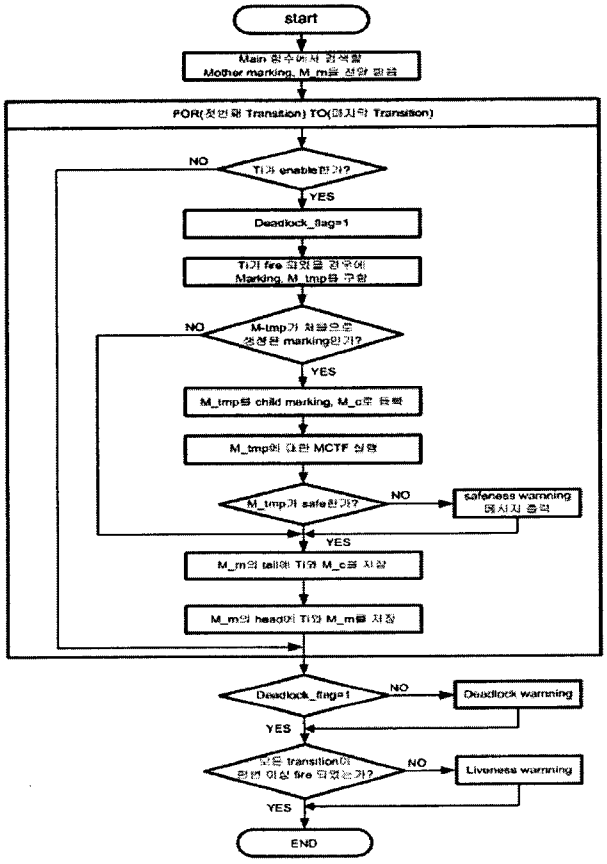


그림 9 PGL 분석가의 트리 생성 및 검사 흐름도
Fig. 9 Tree creating and inspection flow for PGL analyzer

4. PGL을 이용한 작업 프로그램 예제

여기에서는 앞에서 기술한 PGL의 편리성 및 효율성을 보이기 위해 PGL를 이용해 조립 작업프로그램을 수행하는 것에 관하여 기술하였다.

4.1 PGL를 이용한 렌즈의 조립작업

렌즈 조립 시스템의 전체 시스템 구성은 그림 10과 같으며, 렌즈 조립 공정 작업은 그림 11과 같은 작업 공정 순서에 따라 진행 된다.

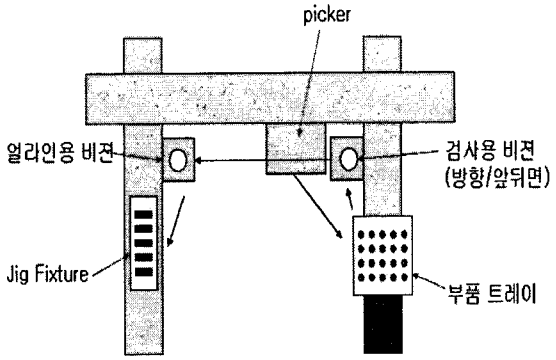


그림 10 렌즈 조립 시스템의 전체 구성도
Fig. 10 Lens assembly system overall architecture

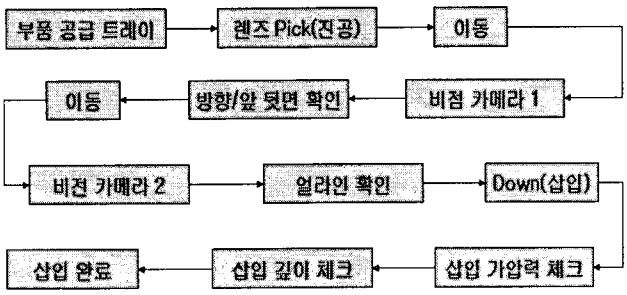


그림 11 렌즈 조립 공정
Fig. 11 Processing of lens assembly

여기서 기술한 조립 작업은 피커(picker)가 부품트레이로 이동하여 렌즈를 진공으로 pick-up해서 이동한 후, 검사 카메라를 이용하여 부품의 상태를 확인하고 베럴(barrel)에 삽입하는 과정을 나타낸 것이다. 여기서 주의할 것은 P_2, P_5, P_9, P_{11}은 다음 작업 경로를 결정하기 위한 조건부 플레이스라는 점이다. 즉 P_2는 공압을 검사하여 피커에 부품이 유무에 따라 다음으로 전이되는 토큰을 T_{14} 또는 T_2로 전달할 것인가를 결정하게 된다. 또한 P_5는 장착할 부품이 양/불에 따라 결정되며, P_9는 위치 보정의 유무에 따라 결정되며, P_{11} 측정되는 반발력의 크기에 따라 토큰의 전달 경로가 결정된다. 이에 대한 PGL 프로그램은 그림 12와 같이 나타나며, 각 플레이스의 의미는 표1과 같고, 생성된 TRM은 표 2와 같다.

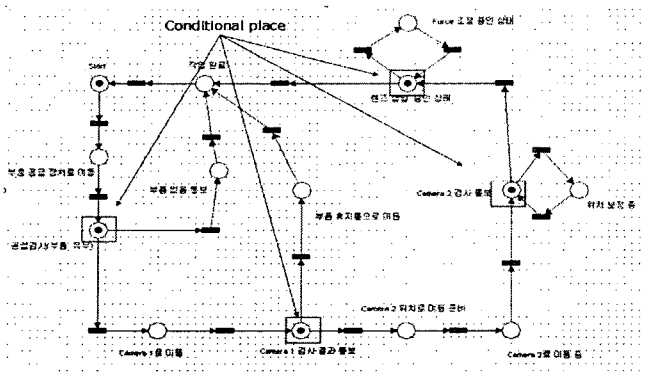


그림 12 PGL를 이용한 렌즈 조립 작업 프로그램
Fig. 12 working instruction program using PGL editor

표 1 렌즈 조립 작업에서 각 플레이스의 의미
Table 1 meaning of each place

	플레이스의 의미		플레이스의 의미
P0	시작 플레이스	P7	Camera 2로 이동 준비(피커 Up)
P1	부품 공급 장치로 이동	P8	Camera 2로 이동 중
P2	피커 공압 검사 결과 통보	P9	Camera 2로 위치 보정
P3	부품 없음 통보	P10	Camera 2에 위치 보정 재 실행
P4	Camera 1으로 이동 중	P11	렌즈 삽입 중 Force 측정
P5	Camera 1 검사 결과 통보	P12	렌즈 삽입 중 Force 재측정
P6	부품 유지통으로 이동 중	P13	작업 결과 통보

표 2 렌즈 조립 작업의 TRM
Table 2 Task Relation Map

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17
P0	-1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
P1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	0	1	-1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0
P3	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P4	0	0	0	1	-1	0	0	0	0	0	0	0	-1	0	0	0	0	0
P5	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0
P6	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0
P7	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	-1	1	0	0
P8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0
P9	0	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	-1	1
P10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1
P11	0	0	0	0	0	0	0	0	1	-1	0	1	0	1	0	0	0	0
P12	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	0	0
P13	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0

또한 PGL 분석기를 통한 분석을 통해 다음과 같은 정보를 얻었다.

```
[1 0 0 0 0 0 0 0 0 0 0 0 0 0] -> T1 ->
[0 1 0 0 0 0 0 0 0 0 0 0 0 0] -> T2 ->
[0 0 1 0 0 0 0 0 0 0 0 0 0 0] -> T3 ->
[0 0 1 0 0 0 0 0 0 0 0 0 0 0] -> T11 ->
[0 0 0 1 0 0 0 0 0 0 0 0 0 0] -> T4 ->
[0 0 0 0 0 0 0 0 0 0 0 0 1 0] -> T12 ->
[0 0 0 0 1 0 0 0 0 0 0 0 0 0] -> T5 ->
[0 0 0 0 1 0 0 0 0 0 0 0 0 0] -> T13 ->
[0 0 0 0 0 0 0 0 0 0 0 1 0 0] -> T10 ->
[0 0 0 0 0 1 0 0 0 0 0 0 0 0] -> T6 ->
[0 0 0 0 0 0 0 0 0 0 0 0 0 1] -> T14 ->
[0 0 0 0 0 0 1 0 0 0 0 0 0 0] -> T7 ->
[0 0 0 0 0 0 0 1 0 0 0 0 0 0] -> T8 ->
[0 0 0 0 0 0 0 1 0 0 0 0 0 0] -> T15 ->
[0 0 0 0 0 0 0 0 0 1 0 0 0 0] -> T9 ->
[0 0 0 0 0 0 0 0 0 1 0 0 0 0] -> T17 ->
[0 0 0 0 0 0 0 0 1 0 0 0 0 0] -> T16 ->
[0 0 0 0 0 0 0 0 0 0 1 0 0 0] -> T18 ->
[0 0 0 0 0 0 0 0 0 1 0 0 0 0]
Reachability Test OK!
```

생성된 도달 가능 트리를 통해 우리는 다음과 같이 작업에 필요한 5개의 간단한 작업 스케줄을 구할 수 있었다.

```
A: {P0 T0 P1 T1 P2 T2 P3 T3 P4 T4 P5 T5 P6
T6 P7 T7 P9 T8 P11 T9  P1}
B: {P0 T0 P1 T1 P2 T2 P3 T3 P4 T12 P13 T13
P11 T9 P1}
C: {P0 T0 P1 T1 P2 T10 P12 T11 P11 T9 P1}
D: {P7 T14 P8 T15 P8}
E: {P9 T16 P10 T17 P10}
```

이를 바탕으로 PGL 번역기를 통해 각 단위 장치에 필요한 작업 파일(job file)을 생성할 수 있다. 렌즈 조립 작업에 필요한 작업 파일은 그림 13과 같은 형태로 작성되었다.

```
INIT:
move_xy feeder_pos
move_xy picker_on
if air_press_flag = off  then
            goto INIT
else
            conitinue
move_xy camera1_pos
v_inspection camera1
if v1_flag = off then
            move_xy wastebasket_pos
            move_xy picker_off
            goto INIT
else
            conitinue
move_z picker_up
move_xy camera2_pos
v_inspection camera2
REPEAT1:
if v2_flag = off then
            move_xy camera2_pos
            move_xy v2_offset
            goto REPEAT1
else
            move_xy v2_offset
move_xy station_pos
REPEAT1:
if force_flag = off then
            move_z up
            rgoto REPEAT2
else
            move_xy picker_off
goto INIT
```

그림 13 작성된 작업 파일
Fig. 13 Generated job command

4.2 OpenGL을 이용한 모의실험

본 논문에서는 생성된 작업 파일의 순차적인 동작 여부를 확인하기 위한 방법으로 OpenGL를 이용한 가상 머신을 3차원 모의 실험기(3D Simulator)로 구현하였다. 구현된 3차원 모의 실험기는 그림 14와 같다. 각 구동기를 보면 jig-fixture move 축과 부품 tray move 축이 Z축을 기준으로 움직이며, 카메라를 내장하고 있는 end-effector 축이 X축을 기준으로 움직여서 lens을 pick-up하고, 카메라를 통하여 부품을 상태를 검사하여 양품이면 barrel에 삽입하기 위하여 Y축을 기준으로 움직이도록 구성 되었다. 그림 15은 동작 중인 모의 실험기이다. 구현된 모의 실험기에서 PGL 컴파일러에서 생성된 직업 파일을 읽어 작성된 작업에 따라 동작을 수행하는 것을 확인할 수 있었다

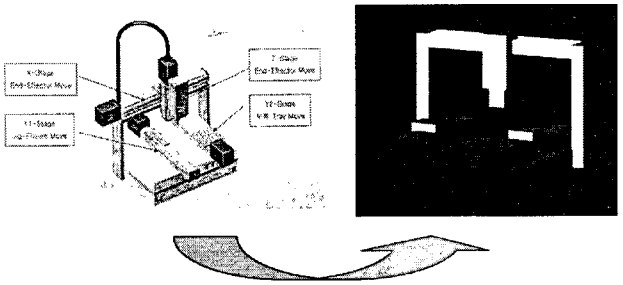


그림 14 모델링 대상인 렌즈 조립 시스템과 가상 머신
Fig. 14 lens assembly system and virtual machine

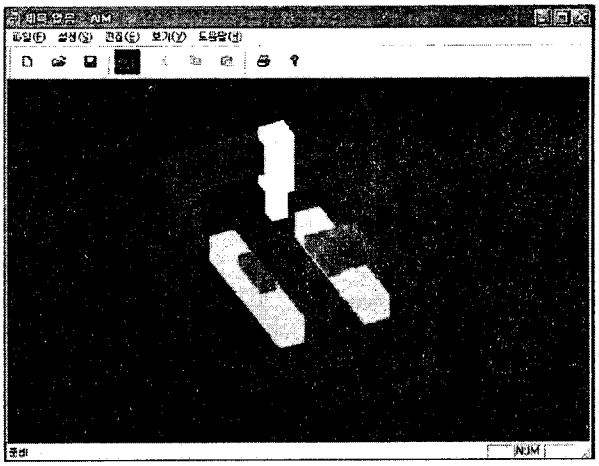


그림 15 3D 모의 실험기
Fig. 15 3D Simulator

5. 결 론

본 논문에서는 관리 제어 시스템의 작업 지시를 위해 페트리네트 기반의 그래픽 언어를 제안하였고, 제안된 제어 언어를 동시성과 동기성을 갖는 여러 가지 작업에 적용하는 예제들을 제시하였다. 또한 제안한 페트리네트 기반 그래픽 제어 언어를 이용하면 여러 개의 장치 모듈들로 구성된 복잡한 시스템에서도 기존의 텍스트 기반 언어 보다 좀 더 표현하기 쉽고, 이해하기 쉬운 장점을 가지고 있다. 본 논문에서 술한 것과 같이 페트리네트의 다양한 해석기법을 도입하여 설계한 PGL 분석기를 개발함으로써 PGL로 표현된 다중 장치 작업에서 일어날 수 있는 논리 오류를 사전에 검색하여 방지할 수 있게 되었다.

향후 남은 과제로는 다양한 페트리네트 해석 기법을 도입하여 응용분야를 확장하는 연구가 필요하며, 보다 효율적인 작업 수행을 위해 작업 스케줄을 최적화 시키는 연구가 필요하다.

감사의 글

본 연구는 산업자원부에서 추진하는 차세대기술개발의 하나로 수행되고 있는 '글로벌 정보공유 및 지식기반 차세대 생산시스템 개발' 과제의 지원을 받아 수행되었습니다.

참 고 문 헌

[1] John. O. Moody, Panos J. Antsaklis, Supervisory Control of Discrete Event Systems Using Petri Nets, Kluwer Academic Publishers, 1998.

[2] J. W. Roach and M. N. Boaz, "Coordinating the Motion of Robot Arms in a Common Workspace," IEEE Journal of Robotics and Automation, Vol. RA-3, No. 5, pp. 437-444, 1987.

[3] M. M. Arbib, R. O. Eason and R. C. Gonzlez,

"Autonomous Robotics Inspection and Manipulation Using Multisensor Feedback," IEEE Trans. on Computer, Vol. 24 No. 4, pp. 17-31, 1991.

[4] H. Chu and H. A. Elmaraghy, "Integration of Task Planning and Motion Control in a Multi-Robot Assembly Workcell," Journal of Robotics and Computer Integrated Manufacturing, Vol. 10, No. 3, 1993.

[5] E. Rutten, et. al., "A Tesk-Level Robot Programming Language and its Reactive Execution," Proc. IEEE Int. Conf. on Robotics and Automation, Vol. 3, pp. 2751-2756, 1992

[6] V. Gopalakrishna, N. Viswanadham and Krishna R. Pattipati, "Sensitivity Analysis of Failure-Prone Flex Manufacturing Systems," IEEE International Conference on Robotics and Automation, Vol. 1, pp. 181-186, 1994

[7] Bertil A. Brandin, "The Real-Time Supervisory Control of an Experimental Manufacturing Cell," IEEE Trans. on Robotics and Automation, Vol. 12, No. 1, pp. 1-14, 1996

[8] I. H. Suh, H. J. Yeo, J. H. Kim, J. S. Ryoo, S. R. Oh, C. W. Lee and B. H. Lee, "Design of a Supervisory Control System for Multiple Robotics Systems," IEEE International Conference on Intelligent Robots and Systems, IROS 96, Vol. 1, pp. 332-339, 1996.

[9] C. W. Moon, B. H. Lee and M. S. Kim, "PLC Based Coordinate Schemes for a Multi-robot System," Proc. IEEE int. Conf. on Robotics and Automation, Vol. 3, pp. 3109-3114, 2001.

[10] E. D. Adamides and D. Bonvin, "Obtaining Synergetic Behavior by Exploiting Relations in Distributed Robot Plans," Proc. IEEE int. Conf. on Robotics and Automation, Vol. 2, pp. 1706-1712, 1994.

[11] F. Y. Wang, et. al., "A Petri-net Coordination Model for an Intelligent Mobile Robot," IEEE Trans. on Systems, Man, and Cybernetics. Vol. 21, No. 4, pp. 777-789, 1994

[12] R. Zurawski and M. C. Zhou, "Petri nets and industrial Applications," IEEE Trans. on Industrial Electronics, Vol. 41, No.6, pp. 567-583, 1994.

[13] L. Ferrarini, C. Maffezzoni and A. Giua, "Design and Imaplementation Issues in The Conrol of Discrete Event Systems," International Conference on Industrial Electronics, Control and Instrumentation, Vol. 3 , pp. 1515-1520, 1994.

[14] I. H. Suh et. al. "A Control System for Multiple-Robot Manipulators; Design and Implementation," Proc. of ISRAM 94, Vol. 5, pp. 279-285, 1994.

[15] Kurt Jensen, Coloured Petri Nets, Springer-Verlag, Chapter 1-4.

[16] MengChu Zhou, Petri Nets in Flexible And Agile Automation, Kluwer Academic Publishers, Chapter 1-6.

[17] J. L. Peterson, Petri net Theory and the modelling of Systems, Englewood Cliffs, NJ, Prentice-Hall, Inc., 1981.

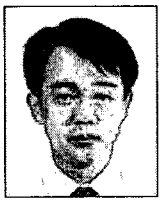
[18] R. Sisto and A. Valenzano, "Mapping Petri nets with inhibitor arcs onto basic LOTOS behavior expressions," IEEE Trans. on Computer, Vol. 44, No. 12, pp. 1361-1370, 1995.

저 자 소 개

공 성 학 (孔 聖 學)

1965년 5월 9일생. 1990년 한양대 전자공학과과 졸업. 1992년 동대학원 전자공학과 졸업(석사). 1992-1997년 LG산전 중앙연구소 선임연구원. 1998년~현재 동대학원 전자통신전파공학과 박사과정

Tel : 82-031-408-5802
Fax : 82-031-408-5803
E-mail : shkong@incorl.hanyang.ac.kr



서 일 홍 (徐 一 弘)

1955년 4월 6일생. 1977년 서울대 전자공학과 졸업. 1982년 한국과학기술원 전기 및 전자공학과 졸업(박사). 1982-1985년 대우중공업 기술연구소 근무. 1985년~현재 한양대학교 정보통신대학원 교수

Tel : 82-02-2220-1080
Fax : 82-2281-3833
E-mail : ihsuh@hanyang.ac.kr