Journal of the Korean Institute of Industrial Engineers
Vol. 25, No. 1, pp. 87-99, March 1999.

# E-Net을 이용한 Heterarchical SFCS 실행 모듈 개발†

홍순도[1] · 조현보[2] · 정무영[2]

[1]LG 반도체(주) / [2]포항공과대학교 산업공학과 · 제품생산기술연구소

# Development of Heterarchical SFCS Execution Module using E-Net†

Soondo Hong · Hyunbo Cho · Mooyoung Jung

A shop floor control system(SFCS) performs the production activities required to fill orders. In order to effectively control these activities, the autonomous agent-based heterarchical shop floor control architecture is adopted where a supervisor does not exist. In this paper, we define functional perspective of the heterarchical shop floor control using planning, scheduling, and execution modules. In particular, we focus on an execution module that can coordinate the planning and scheduling modules and a general execution module that easily can be modified to execute the other equipment. The execution module can be defined informally as a module that downloads and performs a set of scheduled tasks. The execution module is also responsible for identifying and resolving various errors whether they come from hardware or software.

The purpose of this research is to identify all the execution activities and solving techniques under the assumptions of the heterarchical control architecture. And we model the execution module in object-oriented modelling technique for generalization. The execution module modeled in object-oriented concept can be adopted to the other execution module easily. This paper also proposes a classification scheme for execution activities of the heterarchical control architecture. Petri-nets are used as a unified framework for modeling and controlling execution activities. For solving the nonexistence of a supervisor, a negotiation-based solution technique is utilized.

## 1. 서 론

SFCS (Shop Floor Control System)는 비즈니스 시스템에서 입력되어지는 제품 주문서에 따라 제품을 생산한다. 제품 주문서에는 납기 일자, 품질, 생산 우선도 등이 명시되어 있다. SFCS는 명시된 내용에 따라 공정을 실행하기 위하여 생산에 필요한 가공 요소들을 배치하는 역할을 한다. 특히, SFCS는 특정 공정 절차의 선택이나 자원 할당, 작업물의 일정 계획, 가공 명령어 다운로드, 가공의 진척 상황 모니터링, 고장 감지 및 진단, 제조시스템의 상태 보고서 작성 등을 행할 수 있어야 한다. 이렇게 다양한 제어 동작을 일괄적으로 통합한다는 것은 쉽지 않다. 기존의 SFCS 제어 구조 대부분은 계층적(Hierarchical) 제어 모델이 적용되고 있는것이 일반적인데, 계층적 제어 모델에서는 생산에 필요한 모든 활동들이 상위 단계(Supervisor)에서 계획, 스케줄링되어 하위 단계로 지시된다. 이러한 계층적 제어 모델에서는 중간 계층에 있는 어떤 하나의

† 이 논문은 한국학술진흥재단의 공모과제 연구비의 일부지원으로 연구되었음.

모듈이 작동하지 않으면 그 하위에 있는 모든 관련 시스템은 사용할 수 없게 되는 단점을 안고 있어 시스템을 통합하는 데에 어려움이 따르게 된다[1, 3, 4, 6, 7].

본 논문에서는 생산 시스템의 각 구성 요소를 독립적인 단위로 구성하는 Heterarchical 제어 모델을 적용하여 생산 시스템의 목적 즉, 제품을 적기에 생산해 내기 위하여 각 구성 요소들 간의 대화 및 협상에 의해서 생산 활동이 이루어지는 것을 바탕으로 한다. 계층적 제어 구조에서는 단계적으로 제어 문제를 분산시켜 통신량이나 계산량을 줄일 수 있는 반면에 Heterarchical 제어 구조는 통신량과 계산량이 많기 때문에 외면되어 왔으나 컴퓨팅과 컴퓨터 통신 기술의 발달은 Heterarchical 제어 구조의 새로운 가능성을 제시하였다[1].

본 논문의 목적은 생산 현장의 구성 요소들 중에서 직접 가공을 담당하는 가공 기기들의 제어를 위해 필요한 세 가지의 주요 기능(Planning, Scheduling, Execution) 중 Execution(실행) 기능을 담당하는 Executor 즉, 실행 모듈을 개발하는 데 있다.

대상이 되는 실행 모듈을 개발하는 데 있어 (1) 이식성이 뛰어 나고 일반화하기 위해서 객체 지향 모델링 방법을 이용하고 (2) 중요한 요소인 메시지와 Activity의 관계 처리와 메시지 교환시 발생하는 동기화 문제 처리를 위해서 페트리네트(Petri Net) 모델링 방법을 이용한다. 또한, 모델링되어진 결과를 검증하기 위하여 밀링머신, 로봇, 컨베이어, AGV, AS/RS로 구성된 PosTROL 시스템에 대하여 구현해 보고, 동일한 제어 개념으로 모델링되어진 Scheduling 모듈 및 Planning 모듈과의 상호 작용을 점검해 본다[1].

## 2. 기존 연구의 고찰

실행 모듈에 관련된 연구들은 장비 제어를 주기능으로 하여 모니터링, 고장 감지 및 복구의 기능을 추가하며 다양하게 시도되고 있다. 초기의 실행 모듈은 컨트롤러라는 용어로 정의되었으며, 단순한 모니터링이 주기능이었다[9]. 일부 연구에서는 Regulation 또는 Control 등의 기능으로 정의되기도 했다[7, 8].

특히, Joshi *et al.* [8]은 제어라는 용어가 단순히 물리적 제어 이상의 의미를 지니고 있기 때문에 Execution(실행)이란 용어를 사용하였다.

이러한 실행 모듈에 관한 연구는 실행 모듈을 포함하는 Shop Floor Control, FMS Control과 함께 이루어진다. Cho [3] 는 실행 모듈에 중점을 둔 SFCS과 FMS에 관련된 연구들(Huang and Chang [5], Smith and Joshi [15], Huvenoit *et al.* [6], and Cho [3])에 대하여 모델링 방법과 모델링 장비에 따라 정리하였는데, 제어 구조, 모델링 방법, 모델링 장비, Activity 구분, Modelling Level, Combination 작업 가능 여부, 동기 고려 여부에 따라 이를 분류하였다. 그러나, 실행 모듈을 언급하는 SFCS 연구는 대부분 계층적 제어 구조를 따랐으며 Heterarchical SFCS에 관련된 연구들도 시뮬레이션 차원에서 SFCS를 제안하였을 뿐 SFCS를 구현하기 위하여 반드시 필요한 실행 모듈을 언급하지는 않았다.

실행 모듈의 모델링 방법으로는 상태 전이도(State-Transition Diagram)와 페트리네트(Huang and Chang[5], Huvenoit *et al.* [6], Merabet [12], Naylor [13])가 주로 사용되었다.

Huang and Chang [5]은 CTPN(Colored-Timed Petri Net)을 이용하여 에러 감지 및 복구를 할 수 있는 제어 구조를 제안하였으며, Cho [3]는 AND/OR Net와 유사한 실행 그래프를 정의하고 이를 이용하였다. Smith and Joshi [15]는 Material Processor(MP)와 Material Handler(MT)에 대하여 각각의 Formal Model을 상태 전이도(State-Transition Diagram)를 이용하여 제시하였으며 Workstation 단위로 이루어지는 다수의 가공 장비(MP)와 한 대의 로봇(MT) 간의 동기화 작업에 대하여 물리적 모델을 제시하였다.

또한, 물리적 모델에 대하여 Message-based Part State Graph(MPSG)로써 시스템 모델을 구축하였다. Huvenoit *et al.* [6]은 부품의 가공 정보와 생산 자원을 페트리네트를 이용하여 모델링하고 특히, 컨베이어의 모델링과 로봇의 End Effect Change의 모델링에 초점을 맞추었다.

이러한 각 연구는 Shop의 실정에 맞추어 모델링이 이루어졌는데, 모델링 대상이 되는 장비도 다르고, 장비를 지칭하는 용어도 다르다. Smith and Joshi [15]는 Material Processor(머시닝 센터, 자동 선반, 자동 밀링), Material Handler(로봇), Material Transfer(컨베이어), Automated Storage(AS/RS)로 구분하였으며, Huvenoit *et al.* [6]은 Simple Resource와 Complex Resource로 구분하여 모델링하였다. 장비를 정의하는 용어가 다르고 메시지, 즉 한 장비가 행하는 기본 동작의 구분도 다르다. Material Processor와 Material Handler의 경우, tool change, load, process, unload, end effect change, get, move, put으로 세분화할 수 있다. 각 연구의 메시지 단위를 비교 정리한 것이 <표 1>에 나타나 있다.

세분화된 메시지나 Activity가 정해진 순서에 따라 순차적인 처리만이 가능한 경우가 있고, 각 메시지나 Activity를 조합하여 새로운 형태의 작업이 가능한 경우도 있다.

Cho [3]의 연구는 조합 동작이 가능한 경우로서, 예를 들어,

표 1. 메시지 단위의 비교

| | | Huang and Change [5] | Cho [3] | Smith and Joshi [15] | Huvenoit *et al.* [6] |
|---|---|---|---|---|---|
| Material Processor | tool change | change_tool | | | |
| | load | clamp | load | pick | |
| | process | process | process | process | |
| | unload | release | unload | put | |
| Material Handler | end effect change | | c_grip | | grip change |
| | get | clamp | get | mp_get | transfer |
| | move | move | move | | |
| | put | release | put | mp_put | |

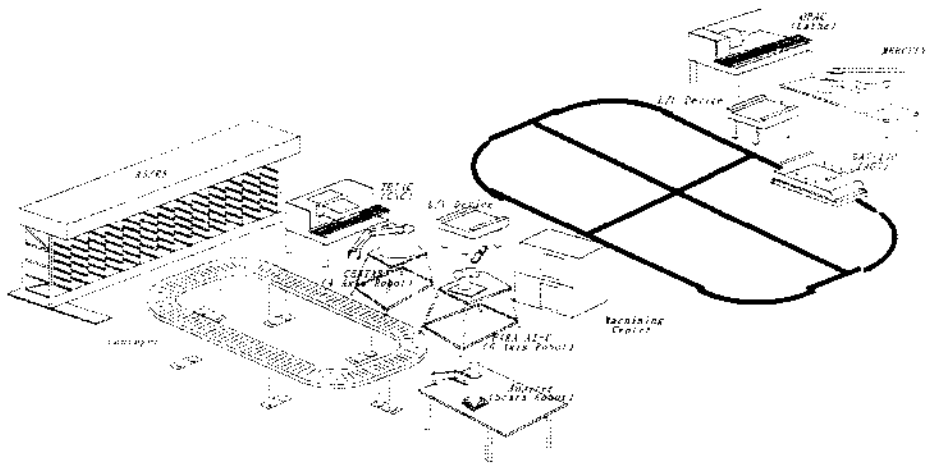


그림 1. PostTROL 시스템의 개관.

로봇의 경우 move 동작 후에 다시 move의 동작을 수행할 수 있다. 실행 모듈이 조합된 동작을 수행할 수 있으면 Decision Making 모듈(Scheduling 모듈, Planning 모듈)은 실시간 On-line 작업시보다 유연한 계획이 가능해 진다는 장점이 있다.

부품을 이송하는 경우에 두 장비의 동기화 작업이 발생하는 경우와 그렇지 않은 경우가 있게 된다. Huang and Chang [5]과 Smith and Joshi [15]는 모든 작업을 동기화 작업으로 간주하고 한 레벨 위의 Workstation 컨트롤러에서 동기화를 조절하였다. Cho [3]와 Huvenoit *et al.* [6]은 Equipment 레벨에서 동기화 작업과 비동기 작업을 구분하여 실행할 수 있게 하였다.

이상을 정리하면 실행 모듈은 다음과 같은 조건을 만족하여야 한다: (1) 조합적인 메시지 처리가 가능하여야 하며, (2) 동기화 작업과 비동기 작업을 구분하여 실행할 수 있어야 하며, (3) 기존 연구에서의 메시지들은 Material Handler와 Material Processor에 대하여 각각 다른 메시지를 정의하였으나 이보다는 메시지를 단순화, 일반화시키는 것이 바람직하다.

실행 모듈의 정의나 모델링에 관계된 연구는 SFCS의 개발시 필연적으로 이루어지는 과정이다. 그러나 Heterarchical 제어 구조에서는 아직 실행 모듈을 구현하지 않고 의사 결정 과정이나 이론에만 머무르고 있다[11].

또한, 기존 연구에서는 실행 모듈이 실제 장비를 제어하여야 함에도 불구하고 다양한 장비 제어에 대한 언급이 없으며 이는 다양한 장비 제어가 용이하지 않기 때문이다. 즉, 장비마다 제어 부분을 서로 다르게 구성해야 하는 문제가 발생하기 때문이다.

그러나 실제로는 장비들의 실행 모듈의 구성이나 장비 제어 방법을 분석하면 해결책을 찾아낼 수 있다. 따라서 본 연구에서는 Heterarchical 제어 구조에 적합한 실행 모듈을 개발하는 모델링 방법으로 객체 지향 방법과 페트리네트를 활용하고자 한다.

## 3. Heterarchical SFCS와 E-Net

본 논문에서는 <그림 1>과 같은 PosTROL 시스템을 그 대상

표 2. PosTROL 시스템 장비들의 분류

| Types | Equipment |
|---|---|
| Material Processor | Orac, Triac |
| Material Handler | Centari, FARA A1-U, Mercury |
| Material Transfer | DAV 100, Conveyor |
| Automated Storage | AS/RS |
| Assembly Machine | Jupiter |

으로 한다. PosTROL 시스템은 Orac(자동 선반), Triac(자동 밀링 머신), Jupiter(스카라로봇), Centari(4축 다관절 로봇), Mercury (이송 로봇), FARA A1-U (6축 다관절 로봇), DAV 100(AGV), 컨베이어, AS/RS로 구성된 SFCS이다. 각 장비는 기능에 따라 Material Processor(부품 가공), Material Handler(부품 이송), Material Transfer(부품 이동), Automated Storage (부품 저장), Assembly Machine(부품 조립)으로 구분된다. PosTROL 시스템의 장비들을 이에 따라 분류한 결과가 <표 2>에 나타나 있다.

### 3.1 SFCS의 정의

PosTROL 시스템의 SFCS를 IDEFO 모델링으로 표현해 보면 <그림 2>와 같다. 시스템 입력(Input)은 원자재(Raw Material)이며 Process Plan, Business Information, Engineering Data 등을 제어 정보로써 이용한다. 사용되는 메커니즘은 머시닝 센터, 자동 선반, 자동 밀링, 이송 로봇, 조립 로봇, 컨베이어, AGV, AS/RS 등의 생산 자원들이다. 제어 정보에 따라 원자재를 생산 자원이라는 메커니즘으로 가공하여 완성된 제품을 생산하게 된다.

본 논문에서 정의하는 Heterarchical SFCS는 계층적 제어 구조에 존재하는 주종 관계를 없애고, 통신을 통하여 시스템의 목적을 달성하는데, 모든 종속 시스템은 자원에 대한 동등한 접근권을 가지며, 시스템간에 상호 접근이 가능하다. 또한, 작업이 서로 독립적이며 전체 시스템의 프로토콜이 엄격하게 지켜진다.

이러한 Heterarchical 제어 모델을 구축하기 위해서는 생산시스템을 구성하는 Autonomous Agent들을 정의하여야 한다. Agent는 의사 결정을 내릴 수 있는 적절한 단위로서, 한 실행 Unit에 대해 하나의 Agent가 존재한다. 이러한 Agent들이 의미하는 바는 고정적인 것이 아니며 본 논문에서의 Heterarchical SFCS는 <그림 3>에서와 같이 Material Processor, Material Handler, Material Transfer, Automated Storage, InitMon으로 구성되어 있다.

### 3.2 Agent의 기능 구조

InitMon을 제외한 각각의 Agent에 대해서는 <그림 4>에 나타난 바와 같이 Planning 모듈, Scheduling 모듈, Execution(실행) 모듈로 역할을 나누어 수행한다. 이들 각각의 모듈에 대해서는 참고 문헌 [1]에 설명이 되어 있으므로 생략하고 다만, 본 논문의 주제인 실행 모듈에 대해서만 간략히 설명한다.

실행 모듈의 역할을 전체적으로 정리한 것을 <표 3>에 나타내었다. 실행 모듈은 우선 메시지를 해석하고 모니터링하여야 하는데, Decision Making 모듈이나 다른 Agent의 실행 모듈에서 전송된 메시지를 분석하여 세부 동작(Activity)을 수행하

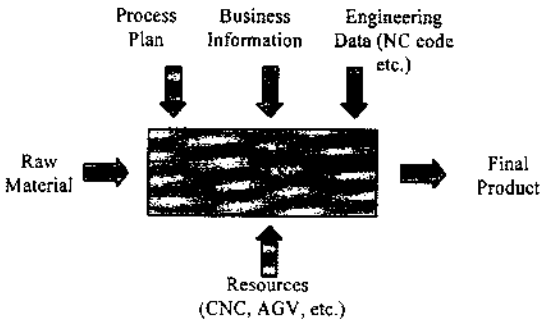



그림 2. SFCS의 Function Process 모델링.

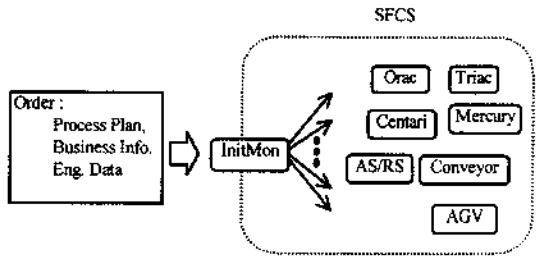



그림 3. Heterarchical SFCS의 Agent 구성 및 관계.

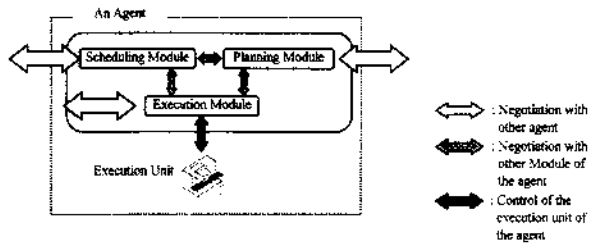



그림 4. Agent의 기능 구조.

표 3. 실행 모듈의 역할

| Item | Examples |
|---|---|
| message interpretation & monitoring | message from decision making 모듈<br>feedback from the execution unit |
| execution-based decision making | message decomposition<br>update the job status board<br>generate job report |
| message broadcasting | reply for the status inquiry |
| control the execution unit | download messages to the execution unit |
| error detection & handling | execution unit error detection & handling<br>communication error detection & handling |

기 위한 실행 베이스 의사 결정을 한다. 그리고 실행 모듈의 상태를 다른 Agent에 알린다. 실행 모듈의 제어 대상이 되는 Execution(실행) Unit는 장비에 따라 여러 제어 방식을 택하고 있다. 실행 모듈은 실행 Unit의 제어방식에 구애를 받지 않고 제어할 수 있어야 하며, 부품의 낙하, 공구의 파손, 통신 장애 등의 에러 상황을 감지하고 이를 진단할 수 있어야 한다.

### 3.3 Execution-Net(E-Net)의 정의

본 논문에서는 메시지를 받아 작용하는 실행 모듈의 동작을 페트리네트를 이용하여 모델링하여 실행 지식으로 이용한다. 이를 위해 Lee [10]의 P-Net를 기본으로 9-tuple E-Net을 아래와 같이 정의한다(<그림 5> 참조).

$$E-Net = \langle P, T, I, O, Cd, M_{Cd\rightarrow PCd}, Act, M_{Act\rightarrow PAct}, M\rangle$$

a) $P = P_{Act} \bigcup P_{Cd}$,
$P_{Act} = \{P_{a1}, P_{a2}, \ldots, P_{an}\}$,

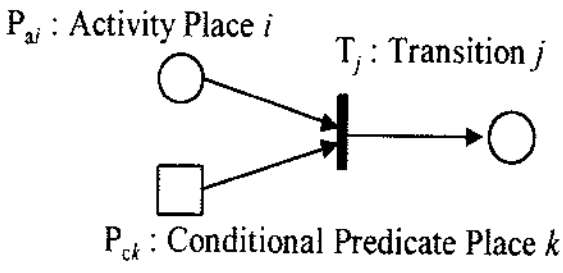



그림 5. E-Net의 표기.

$P_{Cd} = \{P_{c1}, P_{c2}, \ldots, P_{Pm}\}$.

여기서 $P_{Act}$는 activity를 나타내며, $P_{Cd}$는 conditional predicates 를 의미한다. Activity 플레이스는 원으로, conditional predicate 플레이스는 사각형으로 표기한다.

b) $T=\{t_1, \ldots, t_m\}$, $P \bigcup T \neq \emptyset$, $P \bigcap T = \emptyset$.
트랜지션은 production 룰과 같으며, 트랜지션의 점화는 이와 연관된 production 룰의 실행과 일치한다.

c) $I : P * T \rightarrow \{0, 1\}$, $P$에서 $T$로의 direct arc의 집합을 정의하는 함수이며, $O : T * P \rightarrow \{0, 1\}$, $T$에서 $P$로의 direct arc의 집합을 정의하는 함수이다.

d) $Cd$ : Conditional predicates의 집합이다.

e) $M_{Cd\rightarrow PCd}$: $Cd \rightarrow P_{Cd}$, Conditional predicates에서 Predicates 플레이스로의 할당 함수이다.

f) $Act$ : Activity들의 집합이다.

g) $M_{Act\rightarrow PAct}$ : $Act \rightarrow P_{Act}$, Activity로부터 activity 플레이스로의 할당 함수이다.

h) $M = \{M_{Act} \bigcup M_{Cd}\} \rightarrow \{0, 1\}$, $M_{Act}$는 activity 플레이스의 marking을 나타내며, $M_{Cd}$는 conditional predicates 플레이스의 marking을 의미한다.

여기서, 실행 모듈과 관련된 정보는 데이터 베이스 (E라고 정의함)에 따로 저장되어진다.

## 4. 실행 모듈 모델링

기존 연구들이 계층적 제어 구조하에서 Material Processor, Material Handler에 한하여 모델을 제시하였으나, 이 모델들은 Heterarchical 제어 구조나 다양하게 변화되는 실행 Unit의 제

어에는 부적합하다. 따라서 Heterarchical 제어 구조에 적합한 실행 모듈의 매커니즘을 E-Net으로 모델링하고, 다양한 실행 Unit를 모듈 방식 (Modular)으로 용이하게 접근(Accessible)할 수 있도록 실행 모듈간의 관계를 객체 지향적으로 모델링한다.

### 4.1 실행 모듈의 Architecture

먼저 실행 모듈 기능을 5가지로 정의하였는데, 5가지 기능을 수행하는 실행 모듈은 <그림 6>과 같은 지식 베이스 시스템 구조를 갖는다.

일반적으로 지식 베이스 시스템은 지식 베이스(Knowledge Base)와 추론 엔진(Inference Engine), 데이터 베이스(Database)로 구성되지만 실행 모듈은 모델링된 페트리네트를 지식 베이스의 지식으로 사용한다.

따라서 추론 엔진은 페트리네트를 점화하는 점화 기관이 되며 실행 데이터 베이스는 실행 Unit의 정보나 공정 계획 정보, SFCS내의 Agent의 IP Address 등을 저장한다. 추가적으로 통신에 관련된 기능을 수행하는 보조 기능이 필요하며, 실행

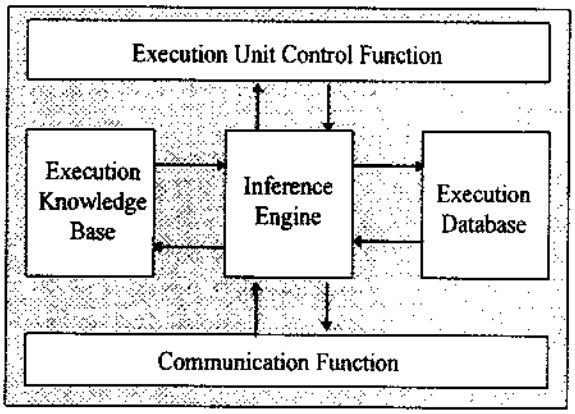



그림 6. 실행 모듈의 Architecture.

표 4. Decision Making 모듈로부터 실행 모듈에 전달되는 메시지

| Message | type | Parameter | Description |
|---|---|---|---|
| process | Control | pid, loc or planid | process pid using planid or Move pid to loc |
| eu_status | Control | void | inquire execution unit status |
| get | Control | pid, null or loc | get pid or get pid from loc |
| put | Control | pid, null or loc | put pid or put pid to loc |
| c_ef_t | Control | tid or efid | change tool in tid or change end effect in efid |

표 5. 실행 모듈로부터 Decision Making 모듈로 전달되는 메시지

| Message | type | parameter | Description |
|---|---|---|---|
| start_process | information | pid | start processing pid |
| end_process | information | pid | end processing pid |
| start_change | information | tid or efid | start changing tid or start changing efid |
| end_change | information | tid or efid | end changing tid or start changing efid |
| start_get | information | pid | start getting pid |
| end_get | information | pid | end getting pid |
| start_put | information | pid | start putting pid |
| end_put | information | pid | end getting pid |
| busy | information | null | execution unit is busy |
| not_busy | information | null | execution unit is not busy |
| find_error | information | error_code | find error_code |

표 6. 실행 모듈들간의 메시지

| Message | type | Parameter | Description |
|---|---|---|---|
| ready_put | information | pid | ready to put pid |
| finish_put | information | pid | finish putting pid |
| finish_get | information | pid | finish getting pid |

Unit 제어 모듈과 통신 모듈은 각각 실행 Unit의 인터페이스와 다른 모듈 및 Agent 간의 통신을 보조한다. 추론 엔진은 이들을 통하여 실행 Unit이나 다른 모듈 및 Agent와 접속이 가능하게 된다.

### 4.2 메시지 정의

실행 Unit로는 Material Processor, Material Handler, Material Transfer, Automated Storage 등이 존재하는데, 기존 연구에서의 Material Processor는 tool_change, load, process, unload로, Material Handler는 change_end_effect, get, move, put으로 기본 동작을 구분하는 것이 일반적이다. 본 논문에서는 실행 모듈의 상태, 즉 기본 동작을 c_ef_t (Change End Effect or Tool), get (Get or Load), process(Process or Move), put(Put or Unload)으로 일반화시킨다. 객체 지향 기법을 이용하여 기본적인 c_ef_t, get, process, put과 점화 등의 Method를 갖는 기본 클래스를 정의하고, 이로부터 계승 받아 다른 클래스를 정의하는 방식을 취하였다. 계승을 받은 하위 클래스는 오버로딩에 의하여 각 클래스에 적합한 c_ef_t, get, process, put을 취하게 된다. 예를 들면, c_ef_t의 경우에 자동 밀링은 공구 교환 작업을, 로봇은 End Effect 교환 작업을 수행하게 된다.

실행 모듈은 c_ef_t, get, put, process 메시지 외에도 다양한 메시지가 필요하다. 메시지의 송수신 객체를 기준으로 구분해 보면, Decision Making 모듈(Scheduling 혹은 Planning 모듈)로부터 실행 모듈에 전달되는 메시지와 반대로 실행 모듈이 이들에게 보내는 메시지, 그리고 실행 모듈이 다른 Agent의 실행 모듈과 동기화 작업시 필요한 메시지로 분류할 수 있다. 첫째, Decision Making 모듈에서 실행 모듈로 보내는 메시지들은 c_ef_t, process, get, put, eu_status 등이 있다. 이들 메시지의 세부 사항은 <표 4>와 같으며 정보와 제어를 구분하여 정의하였다[2]. 여기서 *pid*는 Part Identification Number, *loc*는 Location, *planid*는 Process Plan Information Identification Number, *tid*는 Tool Identification Number, *efid*는 End Effect Identification Number를 의미한다. 둘째, 실행 모듈이 Decision Making 모듈로 보내는 메시지는 실행 도중에 작업의 진행 상태를 알리는 정보메시지이며 <표 5>와 같이 정의한다. 마지막으로 실행 모듈간의 동기화 작업시 필요한 메시지들은 <표 6>과 같으며 어떤 동작이 준비되었음을 알리는 메시지와 동작완료를 알리는 메시지가 있다.

### 4.3 메시지의 E-Net 구성

정의된 메시지들을 이용하여 필요한 E-Net을 구성하여 보면 <그림 7>과 같다. <그림 7>(a)의 get 메시지는 load, pick, mp_get으로 표현되는 동작이다. Material Processor의 경우, 부품을 Load하기 위하여 고정구에 부품을 고정하는 동작을 의미

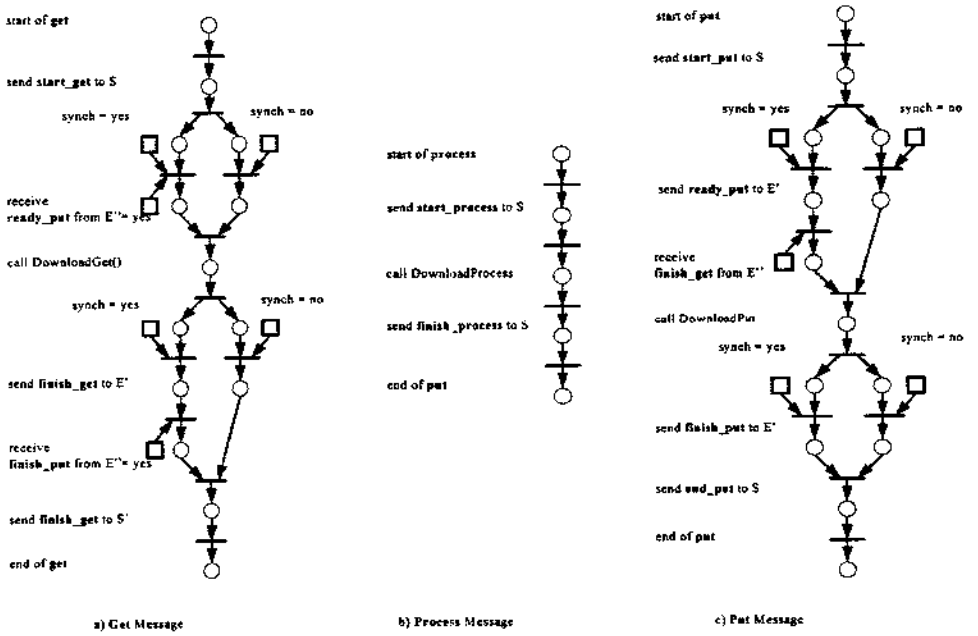



그림 7. 메시지의 E-Net 구성.

한다. Material Handler의 경우는 End Effect를 이용하여 부품을 집는 동작이 된다. AGV나 컨베이어의 경우에는 부품을 AGV 위에 올리거나, 컨베이어 위에 올리게 되는데, 이를 get 동작으로 분석할 수 있다. AS/RS의 경우에는 Hoist가 Hoist Point에서 부품을 집게 되거나, AS/RS에서 부품을 꺼내기 위해 부품을 들어올리는 동작이 get에 해당한다. 한편, process 메시지는 Agent의 주동작으로 <그림 7> (b)와 같이 모델링된다. 머시닝센터, 자동 선반, 자동 밀링 등의 Material Processor의 경우는 G-Code를 Download하여 부품을 가공하는 것을 의미하며 로봇과 같은 Material Handler의 경우에는 이동 동작을 의미한다. 컨베이어와 AGV 역시 이와 비슷하며, AS/RS의 경우는 Rack 사이를 이동하는 경우이다. process 메시지를 받은 후, put이나 get과 같이 Scheduling 모듈에 process의 시작과 종료를 알리게 된다. put 메시지는 <그림 7>(c)에 나타난 바와 같이 모델링되며, unload, put, mp_put 등으로 표현되는 장비들의 동작을 일반화한다. Material Processor의 경우에는 가공이 끝난 부품을 이송 로봇에 전달하기 위해 고정구를 여는 동작이며, Material Handler의 경우에는 End-Effect를 이용하여 집고 있던 부품을 내려놓는 동작이다. AGV, 컨베이어 등의 Material Transfer의 경우에는 이송중인 부품을 이송 완료 후에 내려놓거나 전달하게 되는 경우 put 동작이 발생한다.

AS/RS의 경우에는 부품을 이송 장비에 전달하거나, 저장하기 위하여 Rack으로 가져 간 부품을 내려놓는 경우에 put 동작이 발생한다. 한편, process 메시지의 경우에는 단일 장비내에서 처리되지만, put과 get의 경우는 두 장비의 동기화 (Synchronization) 문제가 발생하게 된다. get과 put을 정의할 때 Conditional Predicate에 의하여 이러한 경우를 구분하였는데 앞서 모델링 되어진 get과 put동작의 ready_put, finish_put, finish_get 의 메시지를 보내고 대기하는 Place를 연결하면 <그림 8>과 같이 get과 put의 동기화 과정이 모델링된다. 비동기 동작은 put 동작이 완료되어야만 get 동작이 시작될 수 있다.

### 4.4 실행 모듈의 객체 지향 모델링

실행 모듈은 실행 Unit에 따라 서로 다른 제어 방법을 택하고 있으며 이 때문에 실행 Unit 마다 실행 모듈을 따로 구현하였다. 그러나, 제어 방법을 제외한 부분들은 앞서 언급한 바처럼 서로 동일한 방법으로서 이루어진다. 따라서 본 연구에서는 장비 의존적인 부분을 제외한 실행 모듈의 요소를 분리시켜 기본 클래스를 먼저 정하고 이로부터 장비 의존적인 부분들을 추가해 나가는 객체 지향 방법으로 실행 모듈을 구현한다. 객체 지향 모델링은 Rumbaugh *et al.* [14]의 정의를 따르고 클래스는 Instance와 Method로서 구성된다. PosTROL 시스템을 구성하는 장비들을 실행 모듈 계승 관계로 표현하면 <그림 9>와 같다. Orac과 Triac의 경우는 Denford 컨트롤러에 의하여 동일한 방법으로 제어된다. AS/RS와 Centari 경우도 829 Robot Servo에 의해 제어된다. 동일한 제어 방법을 따른 경우는 계승(Inheritance) 관계에 의해 실행 모듈을 정의하였다. 계승에 의한 실행 모듈은 기존 장비의 구현뿐만이 아니라, 새로 Shop에 장비가 추가될 경우에도 제어 관련 부분만을 정의, 실행 모듈의 구현이 가능하게 된다.

BasicExecutionModuleClass는 최상위 클래스로 정의되며 장비 의존적이지 않은 요소를 모두 포함하게 된다. <그림 10> (a)에 표현된 바와 같이 Instance로는 E-Net 정보를 저장하는 KnowledgeData, 실행 모듈과 관련된 요소를 저장하는 DataBase Data, 전송될 데이터를 저장하는 SendMessage, 받은 데이터를 저장하는 ReceiveMessage 등이 존재한다. Method로는 E-Net를 점화하기 위한 Firing(), 직렬 통신과 관련된 SendSerialMessage(),

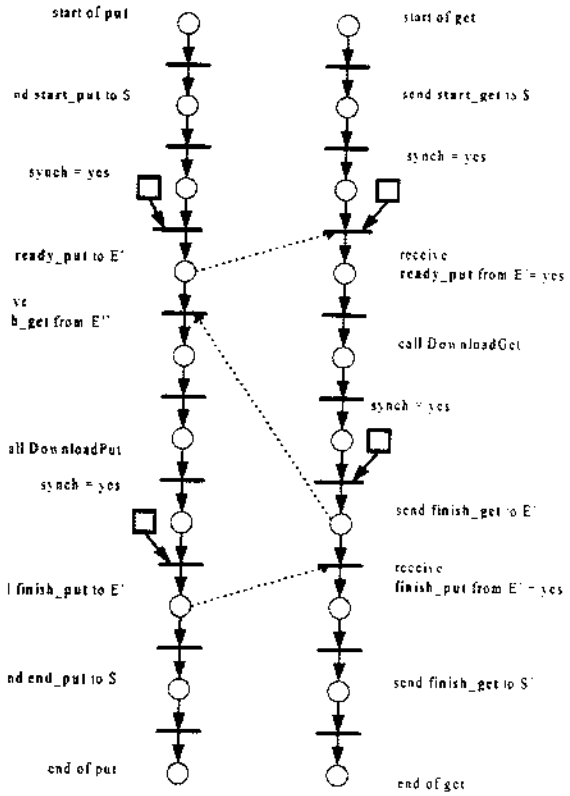



그림 8. 동기화 구성.

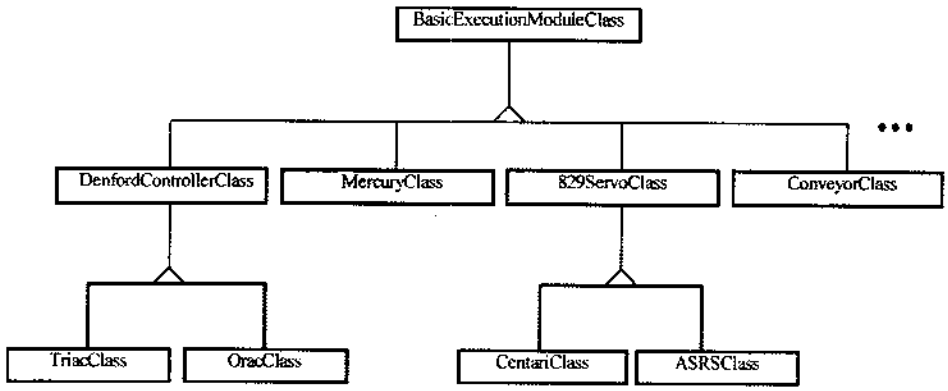
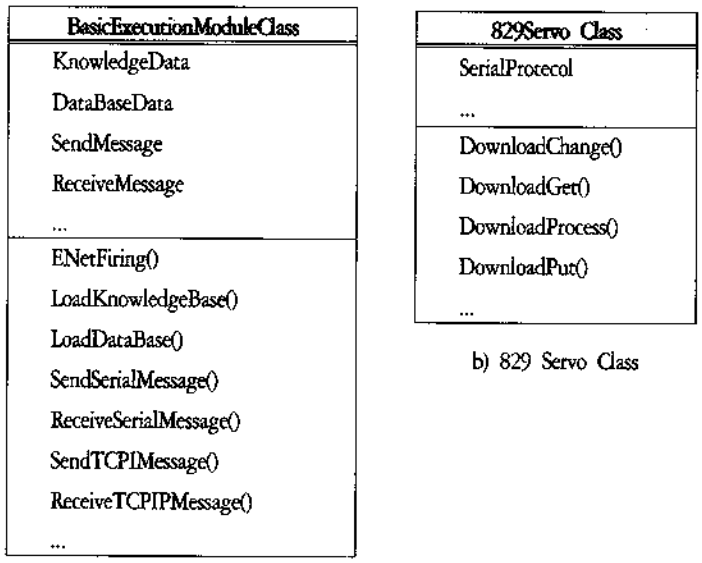



그림 9. 실행 모듈의 클래스 Hierarchy.



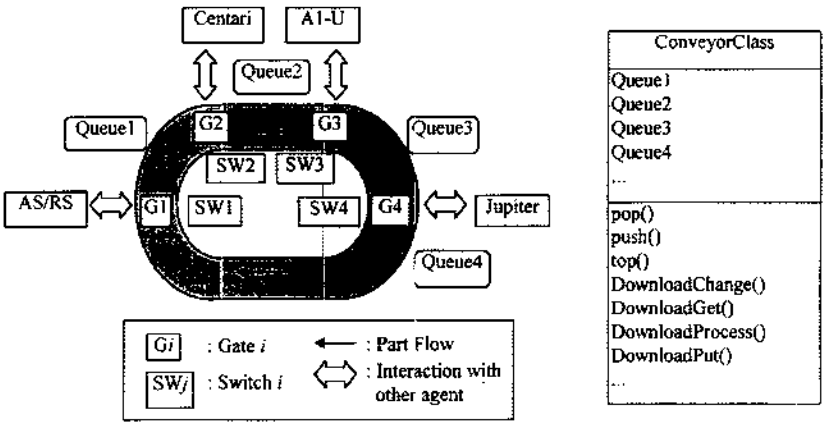

a) Basic ExecutionModuleClass

b) 829 Servo Class

그림 10. 클래스의 예.

ReceiveSerialMessage(), TCP/IP 통신과 관련된 SendTCPIPMessage() 및 ReceiveTCPIPMessage() 가 정의된다. BasicExecutionModuleClass 는 실행 모듈의 최상위 클래스로 일반적인 기능만을 모델링한 것인데, 자동 선반을 이용하여 가공을 한 후에 로봇을 이용하여 부품을 집거나 혹은 컨베이어를 이용하여 부품을 이송하기 위해서는 DownloadChange(), DownloadGet(), DownloadPut(), DownloadProcess()등이 추가되어 져야 한다. DownloadChange(), DownloadGet(), DownloadProcess(), DownloadPut()은 실행 Unit의 하드웨어/소프트웨어 통신 프로토콜에 맞추어 행한다. AS/RS 와 Centari는 829 Robot Servo를 사용하는데, <그림 10> (b)와 같이 829 Servo Class를 정의하고 이로부터 계승 받아 ASRS Class 와 Centari Class를 정의한다.

컨베이어는 <그림 11>(a)와 같이 4개의 스위치와 4개의 Gate로 구성되어 있으며 이송 방향은 시계 방향이다. 컨베이어의 관리는 4개의 Queue에 의해 이루어지는데, Queue1, Queue2, Queue3, Queue4는 Gate (G1, G2, G3, G4)에서 부품이 get되거나 put될 때 재조정되거나 스위치가 ON될 때마다 재조정되기도 한다. Gate에 부품이 get될 때는 를 "if get(pid) in Gi, then push(Queuei , pid)"에 의해 Queue가 재조정된다. 여기서, push(Queuei, pid)는 Queuei에 *pid*를 넣는다. put 동작시에는," if put(pid) in, thenpop(Queue(I+1)mod 4)" 에 의해서 Queue가 관리되며, pop(queuei)는 queuei의 앞의 *pid*를 꺼낸다. 또한, 스위치가 ON될 때마다 Queue를 재조정하게 되는데 를 "if SWi = ON, then push (Queuei, pop (Queue(I-1) mod 4) )"에 의해서 관리된다. 컨베이어의 process 동작은 목적지까지 부품을 이송하는 것이다.




a) 컨베이어 시설 배치 b) 컨베이어 클래스

그림 11. 컨베이어의 시설배치 및 클래스 정의.

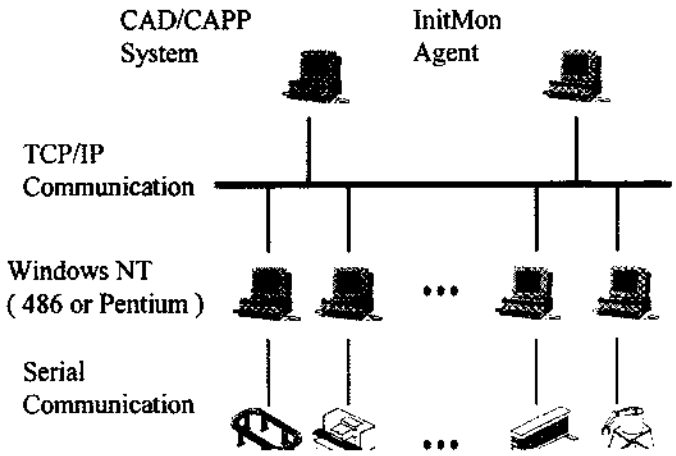



그림 12. PostTROL Network Architecture.

부품을 Garei 까지 이송하는 경우, 다음과 같이 Queue를 검색하면서 대기하는 방법을 취한다.

if process(*pid*) to $G_i$ then
wait until (top ($Queue_i$) = *pid*)
(top($Queue_i$)는 $Queue_i$의 가장 위 부품 번호를 리턴한다)

이상과 같이 컨베이어가 제어되기 때문에 <그림 11> (b)의 ConveyorClass는 Queue를 관리하기 위한 Instance와 Method가 필요하며 DownloadChange(), DownloadGet(), DownloadProcess(), DownloadPut()에 Queue를 재조정하는 기능이 포함된다.

## 5. 실험 및 결과

PosTROL 시스템의 Agent간의 네트워크 구조는 <그림 12>와 같다. CAD/CAPP 시스템은 공정 계획 정보를 생성하고 제품을 주문하며 InitMon Agent는 CAD/CAPP 시스템의 주문을 SFCS에 초기 할당한다. 각 Agent는 TCP/IP를 통한 통신과 Multi-tasking 지원을 위해서 486 PC급 이상의 Windows NT 워크스테이션으로 구축하였다. Agent는 TCP/IP를 통해서 상호간의 정보 교환 및 제어를 하게되며 실행 Unit의 제어 및 상태 정보 교환은 직렬 통신에 의한다. 실행 모듈의 개발은 이상과 같은 환경에서 MS Visual C++ 언어를 이용하였다.

개발된 실행 모듈을 각 Agent에 설치하고 하나의 Agent는 실행 모듈, Planning 모듈, Scheduling 모듈로 구성되어 있다. 각 모듈은 <그림 13>과 같이 하나의 Agent내에 각기 다른 Process로 존재하며 모듈 간에 메시지 교환이 필요한 경우는

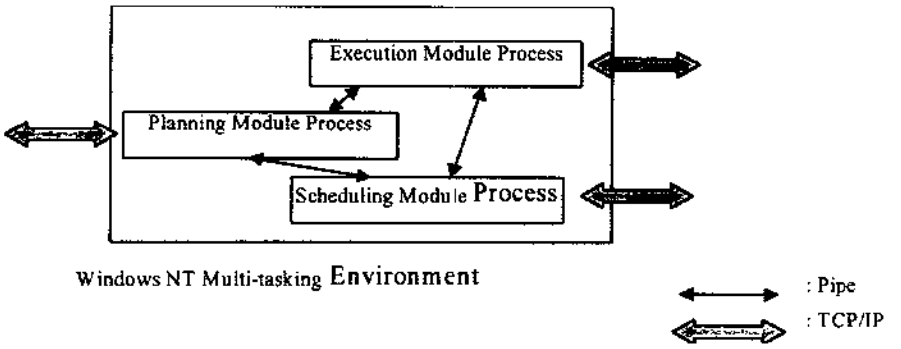



그림 13. Agent의 세 가지 모듈.

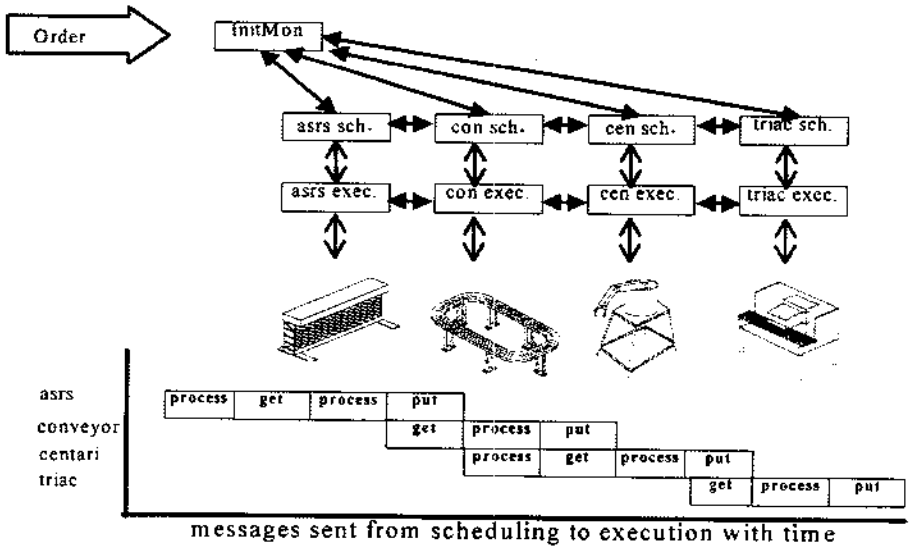



그림 14. Scheduling 모듈과 InitMon 사이의 메시지 예.

Named Pipe 기능을 이용하였다.

실행 모듈은 개별적으로는 어떤 동작도 못하며 단지 Scheduling 모듈, Planning 모듈의 메시지가 있거나, 실행Unit으로부터의 Signal에 의해서 작동하게 된다. 본 논문에서는 <그림 14>와 같은 상황에서 실행 모듈의 시험 동작을 수행하였다. 즉, *Part1*의 Order가 SFCS에 할당되면, InitMon은 공정계획 정보를 이용하여 *Part1*의 가공을 시작한다. InitMon의 가공시작에 관련된 작업 계획은 각 Agent와의 대화를 통해 이루어진다. *Part1*의 경우는 AS/RS에서 원자재를 출고하여 컨베이어와 Centari 로봇에 의해 Triac 자동 밀링에 이송된다. 그런 후, Triac 자동 밀링에서 첫 가공을 하게 되는 공정이 계획된다. 협상의 주체는 부품을 갖고 있는 Agent가 우선한다. 즉, 처음에는 협상의 주체가 InitMon이 되지만 밀링 가공을 마친 후에는 Triac이 주체가 되어 *Part1*과 관련된 다음 협상을 시작한다. AS/RS, 컨베이어, Centari, Triac의 실행 모듈에 <그림 15>와 같이 메시지가 전송되는데, AS/RS에 전송된 첫 메시지 process는 부품을 Load하기 위한 이동이다. 부품을 갖고 있지 않기 때문에 *pid*는 NULL이 된다. *Loc*(L5)는 L5에 대한 교시점을 Return한다. AS/RS의 get과 put은 두 가지가 존재하는데, Rack의 내부에서 get, put하거나 Hoist Point에서 get, put하는 경우가 있으며 실행 모듈에서 교시점의 타입을 추론하여 결정하게 된다. *Loc* ("conveyor") 함수는 컨베이어에 *Part1*를 이송을 위해 교시된 좌표를 찾아 내게 되는데, 교시점이 Hoist Point이므로, 실행 모듈은 Hoist Point에 해당하는 get, put 동작을 한다. 컨베이어에 전달되는 메시지는 get, process, put이다. 그런데 컨베이어의 경우, AS/RS와는 달리 여러 지점에서 부품을 get, put 할 수 있다. 그러므로 get, put의 파라미터에 *Loc*() 함수가 추가된다. AS/RS로부터 부품을 get하는 경우, *Loc*("asrs")가 get 위치를 파악하여 Return한다. put의 경우는 Loc("centari")에 의해 위치를 알아낸다. Centari의 경우는 교시에 의하여 다른 장비의 위치를

process(NULL, *Loc*(5))
get(*part1*)
process(*part1*, *loc*("conveyor""))
put(*part1*)

a) AS/RS

get(*part1*, *Loc*("asrs"))
process(*part1*, *loc*("centari"))
put(*part1*, *loc*("centari"))

b) 컨베이어

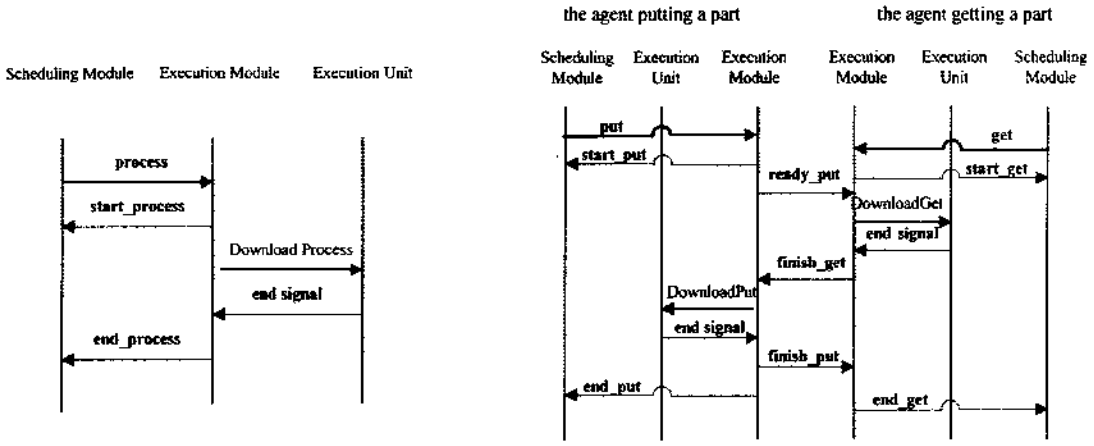
process(NULL, *Loc*("conveyor"))
get(*part1*)
process(*part1*, *loc*("triac"))
put(*part1*)

c) Centari

get(*part1*)
process(*part1*,plan("triac",part1))
put(*part1*)

d) Triac

그림 15. 예제의 경우에 전송되는 메시지.




a) Message Event Diagram when an agent is processing

b) Message Event Diagram when the operations are synchronized

그림 16. Message Event Diagram.

미리 알고 있다. *Loc*("conveyor")는 컨베이어로부터 부품을 get하기 위해 교시된 점을 Return한다. *Loc*("Triac")의 경우는 자동 밀링에 부품을 put하기 위한 교시점을 찾아낸다. 첫 번째 process는 이에 따라 컨베이어와 접촉하기 위한 위치로 로봇을 이동시키고, 두 번째 process는 자동 밀링과 접촉하기 위한 위치로 로봇을 이동시킨다. 부품을 가공하기 위한 자동 밀링이다. get과 밀링 작업을 위해 부품을 고정하고, put은 반대로 고정구를 푼다. process는 실제 가공을 하게 되는데, *plan()* 함수에 의해서 해당 부품을 위한 공정 계획 정보를 찾아낸다.

실행 모듈은 메시지를 입력 받으면 E-Net에 따라 Activity 혹은 Event를 발생시킨다. <그림 16> (a)의 process 경우에 이 process 동작은 실행 Unit 및 Scheduling 모듈과 메시지를 교환하며 동작하였는데, 실선은 제어 메시지, 점선은 정보 메시지를 나타낸다. <그림 16> (b)의 get과 put은 process가 한 실행 모듈의 단독 동작인데 반해 두 Agent가 서로 메시지를 교환하면서 명령을 수행하였다. 메시지 교환은 AS/RS에서 컨베이어로, 컨베이어에서 Centari로, 그리고 Centari에서 Triac으로 부품을 이송할 때 발생하는 동기화 작업시 발생하며 <그림 16> (b)의 Message Event Diagram과 같은 메시지 교환이 발생하였다.

## 6. 결론 및 추후 연구 과제

Autonomous Agent에 기반을 두고 Decision Making 모듈들인 Scheduling 모듈과 Planning 모듈, 그리고 Execution (실행) 모듈로 구성된 Heterarchical SFCS는 모듈성 (Modular), 신뢰성 (Reliable), 신속성 (Fast), 이식성 (Portable), 유연성 (Flexible) 등의 특성을 갖게 되며, 이를 바탕으로 급격한 변화에 대처가 가능하다. 본 논문에서는 이러한 특성을 갖는 SFCS의 제어 모델을 구현하기 위한 실행 모듈의 모델링과 개발을 목적으로 하였다. 실행 모듈은 Decision Making 모듈들로부터의 메시지를 효과적

으로 처리하고, 실행 상황을 Decision Making 모듈들에 모니터링해 주는 것을 주기능으로 한다. 메시지 처리시 다양한 실행 Unit들을 제어하게 되며 실행 Unit들에게서 발생하는 에러 메시지 등을 처리한다.

본 논문에서는, Heterarchical 제어 구조에서의 실행 메커니즘을 효율적으로 모델링하기 위해 E-Net을 제시하였으며, 실행 모듈간의 관계를 객체 지향적으로 모델링하여 다양한 실행 Unit들에 모듈 방식(Modular)으로 용이하게 접근(Accessible)이 가능하도록 하였다. 그리고, 개발된 실행 모듈을 동일 제어 개념으로 개발된 Scheduling 모듈 및 Planning 모듈과 함께 PosTROL 시스템에 적용하여 실험을 수행하였다.

추후 연구 과제로는 : (1) 본 논문에서는 InitMon과 장비의 Agent들만이 존재하는 구조를 정의하였으나 통신 관련 Agent나 시뮬레이션, 생산 자원 Manager와 같이 부하가 큰 부분을 도와주는 Agent의 추가가 필요하며, CAD/CAPP에 관련된 Off-line Agent들과의 관계도 정립되어야 한다. (2) 발생 가능한 에러를 구분하고 각 에러에 맞추어 적합한 처리를 할 수 있는 능력을 갖추어야 한다. (3) 현재 정의된 실행 기능은 Equipment Level에서 사용될 수 있으나 Workstation Level에서의 실행 기능까지도 포함하는 확장된 실행 모듈이 필요하다.

## 감사의 글

본 논문이 보다 나은 내용과 표현을 갖도록 도와주신 두 분의 심사위원들께 감사의 말씀을 드립니다.

## 참고문헌

1. 김화진, 조현보, 정무영, "Heterarchical SFCS를 위한 가공기계의 Planner 모듈 개발," *대한산업공학회지*, 제22권, 제4호, pp. 719-739, 1996.
2. Barkmeyer, E. J., "Some interactions of information and control in integrated automation systems," *Advanced Information Technologies for Industrial Material Flow Systems, NATO ASI Series*, Vol. F53, Springer-Verlag, New York, 1989.
3. Cho, H., *An Intelligent Workstation Controller For Computer Integrated Manufacturing*, Ph.D. Dissertation, Texas A&M, 1993.
4. Fox, B. R. and Kempt, K. G., "Complexity, uncertainty and opportunistic scheduling," *Proceedings of the IEEE Second Conference on Artificial Intelligence Applications: The Engineering of Knowledge-Based Systems*, Miami Beach, Fl, pp. 487-492, 1985.
5. Huang, H. P. and Chang, P. C., "Specification, modeling and control of a flexible manufacturing cell," *International Journal of Production Research*, Vol. 30, No. 11, pp. 2515-2543, 1992.
6. Huvenoit, B., Bourey, J. P. and Craye, E., "Design and implementation methodology based on Petri net formalism of flexible manufacturing systems control," *Production Planning and Control*, Vol. 6, No. 1, pp. 51-64, 1995.
7. Jones, A. T. and Saleh, A., "A multi-level/multi-layer architecture for intelligent shop floor control," *International Journal of Computer Integrated Manufacturing*, Vol. 3, pp. 60-70, 1990.
8. Joshi, S. B., Wysk, R. A. and Jones, A. "A scaleable architecture for CIM shop floor control," *Proceeding of CIMCON '90*, pp. 21-33, 1990.
9. Larin, D. J., "Cell control: What we have, what we will be need," *Manufacturing Engineering*, pp. 41-48, January 1989.
10. Lee, K., *Formal Integration of Process Planning and Shop Floor Control for CIM*, Ph.D. Dissertation, Pohang University of Science and Technology, 1994.
11. Lin, G. Y. and Solberg, J. J., "Integrated shop floor control using autonomous agents," *IIE Transaction*, Vol. 24, No. 3, pp. 57-71, 1992.
12. Merabet, A. A., "Synchronization of operations in a flexible manufacturing cell: The petri-net approach," *Journal of Manufacturing Systems*, Vol. 5, No. 3, pp. 161-169, 1986.
13. Naylor, A. W. and Volz, R. A., "Design of integrated manufacturing system control software," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 17, No. 6, pp. 881-897, 1987.
14. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W., *Object-Oriented Modelling and Design*, Prentice-Hall International, 1991.
15. Smith, J. S. and Joshi, S. B., "Formal models of execution module in shop floor control," *Computer Control of Flexible Manufacturing Systems*, pp. 285-314, 1994.

1998년 2월 접수, 1999년 2월 채택