Journal of the Korean Institute of Industrial Engineers
Vol. 25, No. 2, pp. 150-161, June 1999.

Heterarchical SFCS를 위한 ICPL 개발에 관한 연구

권성필[1] · 조현보[2] · 정무영[2]

[1]삼성 SDS / [2]포항공과대학교 산업공학과 · 제품생산기술연구소

Development of ICPL for Heterarchical SFCS

Sungpil Kwon[1] · Hyunbo Cho[2] · Mooyoung Jung[2]

This paper deals with the design and development of a real-time integrated communication architecture for heterarchical SFCS(Shop Floor Control System). In autonomous agent-based heterarchical SFCS, each functional unit of parts and resources is equipped with an intelligent controller (agent) that acts as the representative of the entity. The controllers communicate and negotiate with other controllers on a real-time basis through message passing and bidding protocol to achieve mutual agreements for task sharing. ICPL(Integrated Communication Protocol and Language) is proposed for this purpose. ICPL is a language and a protocol for supporting communication among intelligent controllers. Based on the speech act theory, this paper proposes a semantic description for ICPL that associates the description of the cognitive states of controllers with the use of language primitives (message_type). Semantics for the basic set of ICPL messages is described. Eventually, an ICPL-based communication architecture can provide the implementation of the distributed and heterarchical SFCS, and makes the intelligent controller transparent to the negotiation problem.

1. 서 론

SFCS의 다양한 제어 동작을 통합하기 위하여 보통 hierarchical architecture 제어 모델이 적용되어져 왔다. Hierarchical 제어 모델에서는 SFCS의 구조를 상하 관계로 구성하고 생산에 필요한 모든 활동들이 상위 단계에서 계획, 관리되어 하위 단계로 지시가 내려간다. 이러한 모델이 널리 사용되어지고 있는 이유는 복잡한 생산 시스템의 구조를 쉽게 이해할 수 있으며 기존의 생산 현장에 대한 적용이 용이하기 때문이다. 또한 시스템을 전체적으로 제어함으로써 시스템 전체의 최적화가 어느 정도 가능하기 때문이다. 그러나 hierarchical 제어 모델은 계층이 존재하기 때문에 전체 시스템 구조의 변화에 신속하게 대처하지 못하며, 한 장비의 고장이나 시스템 일부의 확장이 시스템 전체에 많은 영향을 미치게 되어 신속한 시스템의 개발을 어렵게 한다는 단점을 갖고 있다(Dilts, *et al.*, 1991). 최근에 와서 고객들의 요구로 인해 보다 다양한 제품의 개발 및 생산이 요구되어지고 이로 인해서 제품의 수명 주기도 짧아지고 있다. 또한 변화하는 시장 속에 고객들이 원하는 제품을 적시에 생산, 공급해야만 기업간의 경쟁에서 이길 수 있게 되었다. 따라서 끊임없는 시장 환경의 변화에 신속히 대처하고 경쟁력을 효과적으로 확보하기 위해서는 기존의 생산 시스템 구조인 hierarchical 제어 모델보다는 heterarchical 제어 모델로 변화할 필요가 있다. 기존의 hierarchical 제어 모델로서는 급변하는 시장 환경에 신속하고 유연하게 대처하기 힘들기 때문이다. 또한 제어 기술 및 정보/통신 기술의 급속한 발달로 인해 heterarchical architecture의 구현이 가능해졌다는 것도 이를 추진하는 데에 좋은 환경을 제공하고 있다. 따라서 무엇보다도 민첩성이 요구되는 생산 시스템에는 heterarchical architecture 제어 모델의 적용이 요구되고 있다(Kim, *et al.*, 1995).

본 연구에서는 autonomous agent를 기반으로 하는 heterarchical SFCS을 적용한다. 컴퓨팅과 컴퓨터 통신 기술의 발달은 heterarchical 제어 모델의 새로운 가능성을 제시하였는데 autonomous agent를 기반으로 한 SFCS는 heterarchical 제어 구조의 한 모델이라 볼 수 있다. 이는 계층적 제어 모델과 비교되는 개념으로서 제품을 적기에 생산한다는 생산 시스템의 목적을 달성하기 위해 shop 내의 각 구성 요소(agent)들간의 통신과 협상(negotiation)에 의해서 생산 활동이 이루어지는 제어 모델을 의미한다. Heterarchical 제어 모델을 적용하게 되면 SFCS의 구성 및 기존 system의 재구성이 용이하게 되는데, 이는 현재의

시스템에 새로운 장비들을 도입하는 경우 전체 SFCS를 다시 구성할 필요 없이 필요한 구성 요소(agent) 들을 단순히 끼워 넣기만 함으로써(plug-and-play) 새로운 system을 구성할 수 있다는 의미다(Hong, *et al*, 1995). 그러나 heterarchical 제어 모델의 구현에 있어서 가장 큰 문제점들 중의 하나가 controller들간의 방대한 통신 문제이다(Kramer and Senehi, 1993). 기존의 제어 모델에서는 중요시되지 않았던 자율적인 협상과 통신 문제가 heterarchical 제어 모델에서는 필수적인 요소로 부각되며 이를 해결하기에는 기존의 통신 방식은 부적절한 것이 사실이다. 즉, 이러한 통신 문제를 해결하기 위한 충분한 표현력과 적절한 semantics을 가지고 있는 통신 언어 및 프로토콜이 없다는 것이다(Maturana adn Norrie, 1997). 본 연구에서는 이러한 문제점을 해결하고 이질적인 controller들간의 정보를 교환하기 위해서 표준화된 통신 언어인 ICPL (Integrated Communication Protocol and Language) 을 제안한다. 이는 각 controller 하부에 존재하면서 전체 네트워크와의 연결을 책임지는 ICPL을 기반으로 하는 communication architecture를 개발하는 데 사용될 수 있다. ISO의 Open System Interconnection (OSI)의 7-layer 구조에 부합되게 개발될 것이므로 다양한 생산 환경의 네트워크에서도 개방적이고 분산화된 통신 기능을 수행할 수 있게 된다.

2. 기존 연구의 고찰

SFCS 구조에 관한 연구 중 centralized architecture나 hierarchical architecture에서는 상위 controller로부터 단순히 command를 받고 그 결과를 status report로 보내 주기만 하면 되기 때문에 통신의 역할이 그리 중요시되지 않는다. 그러나 heterarchical architecture에서는 각 controller들은 하나의 작업을 알려서 bid를 요구하고 bid를 만들어 내어 최종적으로 contract가 이루어지는 협상의 기능을 수행해야 하기 때문에 통신의 중요성이 부각되기 시작하였다. 그러나 컴퓨팅 기술과 정보/통신 기술 등이 이에 미치지 못하여 현실적으로 어려움이 많았으나 최근 이 분야 기술의 급속한 발전으로 통신 문제를 해결할 수 있게 되었고, 소프트웨어 분야에서 활발히 진행되고 있는 multi-agent 시스템의 소프트웨어 기술을 생산 분야에 적용시킴으로써 heterarchical architecture의 구현이 가능하게 되었다(Hong, *et al.*, 1995; Maturana and Norrie, 1997).

먼저 communication architecture 분야를 살펴보면, 생산 현장에 가장 잘 알려진 communication architecture는 MAP (Manufacturing Automation Protocol)이다. MAP는 1980년대 초 GM (General Motors)에 의해 제안되어 현재까지 계속 발전되고 있는 중이다. 또한 유럽에서는 CNMA (Communication Network for Manufacturing Applications) 라 불리는 MAP과 유사한 architecture가 개발되어 확장중이다. 사용되는 통신 프로토콜은 ISO의 OSI 모델에 근거한 7-layer 모델을 사용하며 factory-floor-station 들을 연결하는 물리적인 경로와 서비스들을 제공한다. Communication architecture는 각 계층에 따라 필요한 서비스들을 정의하고 있는데 MAP과 CNMA와 같은 communication architecture는 응용계층에 MMS(Manufacturing Message Specification) 서비스를 표준안으로 채택하고 있다. MMS는 공장 환경의 일반적인 통신 환경을 규정하고 있는 핵심 규격이고 공장의 특수한 물리적 장치인 process control, programmable controllers, numerical control 그리고 robot 등의 특정 통신에 대해서는 각각 표준화 기관의 주도로 MMS companion standard가 제정되고 있다. 그러나 MAP와 MMS의 적용성은 SFCS로 운영되어지는 생산 환경에서는 아직도 매우 한정적이다. 기존의 생산 장비들은 MAP와 MMS를 이해할 수 있는 능력이 없으므로 MAP과 MMS 표준안을 지원하지 않는다. MAP와 MMS의 호환 기종들이 개발되어지기는 하지만 과연 소규모의 SFCS에서 비용과 활용성 문제로 인해 적용되어질지는 불투명하다(Messina and Tricomi, 1990; Tiemersma, *et al.*, 1993; Valenzano, *et al.*, 1992).

Agent를 한마디로 정의하긴 힘들지만 분산 환경하에서 자율적으로 작업을 수행하는 지적인 특성을 갖는 응용프로그램으로 정의할 수 있다(Shoham, 1993). Agent에 대한 기존의 연구 방향은 크게 두 가지로 나누어진다. 하나는 단독으로 수행하는 지능적인 기능에 중점을 맞춘 기능성 agent에 대한 연구이고, 다른 하나는 여러 agent들이 존재하는 상황에서 그들 서로간의 상호 작용과 협동에 초점을 맞춘 multi-agent 환경에 대한 연구이다. 이 중 heterarchical architecture와 그 구조 개념이 유사하여 적용이 가능하고 또한 실제 적용되어지고 있는 multi-agent 시스템은 heterarchical architecture의 구현에 하나의 방법을 제시하고 그 가능성을 보여 줌으로써 생산 통합화에 새로운 기준을 가져다 주었다(Maturana and Norrie, 1997). Multi-agent에서 발생하는 중요한 문제는 각 응용 agent의 이형질성(heterogeneity)이다. 물론 multi-agent architecture와 응용 agent들을 처음부터 같이 개발했다면 이러한 문제가 없겠지만 기존의 응용 프로그램들을 바탕으로 agent 기능을 추가한 형태의 agent인 경우는 기존의 응용 프로그램의 특성에 따라 서로 다른 형태를 지니게 된다. 이와 같이 이형질성의 응용 agent간의 통신을 위해서는 표준화된 메시지 형태와 전달 프로토콜이 있어야 한다(ARPA Knowledge Sharing Initiative, 1993).

Multi-agent 시스템에서 가장 큰 특징이면서 중요한 문제인 agent간의 통신 문제를 해결할 수 있는 방법은 공통된 agent 통신 언어인 ACL(Agent Communication Language)을 확보하는 방법이다(ARPA Knowledge Sharing Initiative, 1993). Agent끼리 대화한다는 것은 정해진 언어 규약에 따라 메시지를 주고받음을 의미하며 하나의 agent는 다른 agent에게 서비스를 요청하기 위해 정해진 언어 규약에 따라 요구 사항을 메시지 형태로 바꾼 후 해당 agent에게 전달한다. 다른 agent로부터 서비스 요청을 받은 agent는 그 메시지를 분석해 내부에서 처리할 수 있는 형태로 변환해서 이를 처리한다. Agent는 그 결과를 다시 메시지 형태로 바꿔 이를 요청한 agent에게 전달한다(Shoham, 1993).

이와 같이 agent끼리 대화를 하려면 내부적인 통신 규약이 필요하다. 이러한 multi-agent 시스템 구축과 agent통신 언어 제정에 대한 노력이 '지식공유'문제로 집약되어 많은 연구가 진행되고 있다. 그 중 가장 대표적인 연구 성과로는 ARPA (Advanced Research Projects Agency) 의 Knowledge Sharing Effort에 의해 규약화된 agent간의 통신 언어인 KQML(Knowledge Query and Manipulation Language)이 있다(Finin, 1992). 즉, KQML은 각각의 agent가 서로 다른 개발자에 의해 서로 다른 시기에 서로 다른 platform을 바탕으로 서로 다른 목적을 위해 개발되었기에 발생하는 이형질성을 해결하기 위해 표준화된 메시지 형태와 전달 프로토콜을 제공하는 ACL이다. 그러나 KQML은 agent 상호간의 지식 공유와 지식 전달에 목적을 두고 개발되었기 때문에 SFCS에 바로 적용시키는 것은 불가능하다. 실제로 KQML을 이용하여 적용시켰던 사례들도 KQML을 그대로 적용시킨 것이 아니라 그 환경에 맞게 변경시킨 후 시스템을 독자적으로 개발하였다. 즉, KQML을 적용시켜 시스템을 SFCS에 맞게 개발하는 것보다 SFCS에 적합한 통신언어를 개발한 후 이것을 바탕으로 시스템을 개발하는 본 연구의 의도가 훨씬 효율적이라고 할 수 있다.

한편, agent 상호간의 협의를 가능하게 하는 contract net 프로토콜(Smith, 1980)은 분산 환경하의 문제 해결을 위한 통신과 제어를 명확히 하기 위해 개발되어졌다. 다시 말해서 contract net 프로토콜은 분산 환경하의 문제 해결을 담당하고 있는 노드들간의 상위 계층의 통신 프로토콜이다(David and Smith, 1983). 각 작업을 담당하고 있는 노드들간에 어떻게 작업들을 할당할 것인가 하는 것이 가장 중요한 문제이다. 또한 resource들을 할당하는 문제 역시 분산 환경하에 중요 요소로 대두하는데 이 문제는 multi-stage 협상 프로토콜에서 중점적으로 다루고 있고, contract net 프로토콜은 작업 할당 문제를 중요 요소로 갖고 있다. 하나의 노드가 task announcement를 통해 작업을 다른 노드들에게 알리고 그에 따른 반응들을 bid들을 통해 전달한다. 받아들인 bid들 가운데 best bid를 제시한 노드에게 award를 줌으로써 contract가 이루어지게 된다(Smith, 1980).

3. ICPL (Integrated Communication Protocol and Language)

Heterarchical architecture를 기반으로 하는 시스템에서 controller들끼리의 자율적인 문제 해결 방법으로서 사용되는 통신 방식은 충분한 표현력과 각 controller들끼리의 정보 교환에 필요한 적절한 semantics를 갖추어야 하며 heterarchical architecture의 가장 큰 특성 중의 하나인 자율성(autonomy)에 적합해야 한다. 또한 다른 작업들을 각자 수행하는 이질적인 controller들간의 정보를 교환하기 위해서는 표준화된 통신 언어가 필요하다. 본 연구에서 제시하는 표준화된 통신 언어인 ICPL(Integrated Communication Protocol and Language)은 heterarchical control architecture에 기반을 둔 생산 시스템에서의 자율적인 협의와 통신을 위한 상위 레벨 통신 언어/프로토콜이다. 그 특징을 보면 다음과 같다:

(1) 정보 교환에 관련된 통신 상황에 중점을 둔 통신 언어/프로토콜이다.
(2) 통신의 내용이 되는 정보의 형태나 의미에 독립적이다.
(3) 자율적(autonomous)이고 비동기적(asynchronous) 통신을 가능하게 한다.
(4) 쉽게 기존의 controller들에 wrapping되어 사용되어질 수 있다.
(5) 단순한 구문 위주의 프로토콜 이상이다.

이 중에서 (5)의 의미는 중요한데 이는 ICPL이 단순한 통신 프로토콜의 구문 정도가 아닌 자신을 이용할 통신 상황에 대한 이해와 이에 기반한 heterarchical SFCS의 구조에 대한 이해도 포함하고 있음을 의미한다. 실제 뒤에서 살펴볼 ICPL에 대한 사양에는 ICPL이 단순한 통신 프로토콜 정도를 넘어, 자율적 생산 시스템 전반에 대한 semantics와 이를 이용하는 controller 구조에 대해서도 제시하고 있음을 알 수 있다.

3.1 ICPL string syntax

ICPL은 message_type에 따라 분류되어지는데 message_type은 본 내용에서 정의되어지는 syntax를 사용하여 ASCII string으로 표현되어진다. Syntax의 대체적인 모습은 multi-agent 시스템에서 agent 사이의 통신 언어인 KQML의 syntax를 기본으로 하여 message_type name과 parameter list로 이루어진다(Finin, 1992). message_type은 reserved message_type으로 구성되어 있고 parameter list는 parameter name과 parameter value의 쌍들로 이루어진 list이다. Parameter name으로 불려여지는 keyword들은 colon(:)으로 시작되어야 하고 parameter value의 앞에 있어야 한다. keyword들로써 구분되어지는 parameter들은 순서에 상관없이 keyword들로써 인식되어진다. Context free form인 BNF로써 표현되어지는 ICPL string syntax는 <그림 1> 에서 보는 바와 같으며 <alphabetic>, <numeric>, <ascii>, <whitespace>들은 정의가 되어 있다고 가정한다.

3.2 Framework for the semantics of ICPL

ICPL은 한마디로 자율적인 생산 시스템에서 controller들간의 run-time 정보 공유 및 전달과 상호 작용을 지원하는 메시지 처리 프로토콜이라고 할 수 있다. 여기서 상호 작용이란 단순히 메시지를 주고받는 이상을 의미하며 ICPL은 명확한 언어적인 행위들을 통해서 실제 정보 교환 과정을 모델링하게 된다. Controller들이 이용하는 통신 언어를 모델링함에 있어서 이러한 상호작용에 적합한 semantics를 마련하는 것은 무엇보다 우

```
<message_type>          ::=          <word>{<whitespace>          :
<word><whitespace><word>}
       <whitespace> : <word><whitespace><expr>
<word> ::= <char>{<char>} | <kword> | <rword>
<char> ::= <alphabetic> | <numeric> | <special> | <ascii>
<special> ::= < | > | ? | / | { | } | [ | ] | \ | = | + | - | _
       | ) | ( | * | & | ^ | % | $ | # | @ | ! | ~ | `
<kword> ::= RECEIVER | SENDER | ID | CM | CONTENT
<rword> ::= ADVERTISE | ANSWER | AWARD | BROKER |
            ERROR | MONITOR | QUERY | RECOMMEND |
            REGISTER | REQUEST | SORRY | TELL | UNREGISTER
```

그림 1. ICPL string syntax in BNF.

선되어야 한다. 적합한 semantics를 갖추기 위해선 이론적인 배경에 근거하는 formal한 분석 방법과 접근 방법이 필요하다 (Singh, 1993).

ICPL을 위한 Speech Act Semantics

Speech Act Semantics는 사람들 사이의 의사 전달을 고려하여 언어 학자들이 발전시킨 상위 레벨의 이론적인 framework이다 (Searle, 1969). 이것은 임의의 agent들의 통신을 위한 일반적인 모델로서 Computational Linguistics나 AI 분야에서 적용되어 왔다(Labrou, 1996). 이러한 Speech Act Semantics는 SFCS에서 controller들간의 통신을 위한 ICPL에서도 semantics를 위한 framework을 제공해 줄 수 있다. Speech Act Semantics는 말 그대로 언어를 주로 행위적 측면으로 보는 것인데 다음의 세 가지 action들이 있다(Labrou, 1996):

(1) Locution - 물리적으로 존재하는 말 그 자체
(2) Illocution - 물리적인 말 이외에 receiver에게 전해지는 sender의 의도
(3) Perlocution - illocution의 효과로 인해 발생하는 action.

예를 들어 '부품을 가공하라!' 는 locution은 부품을 가공하라고 하는 명령의 전달인 illocution과 receiver가 실제 부품을 가공하는 행위인 perlocution을 수반한다고 할 수 있다. 그러나 어떤 Speech Act를 고려하든지 controller 통신 언어를 설계함에 있어서 heterarchical SFCS의 특징에 적합한 controller의 인지 상태 (cognitive state)에 대해 참조할 수 있도록 만드는 작업은 필수적이다. ICPL의 semantics를 정의함에 있어 본 연구에서는 Searle (1969)이 기술한 Speech Act Theory를 적용한다.

ICPL의 Semantics를 위한 기본 구조

ICPL의 semantics 구현을 위한 기본 구조는 먼저 controller들간의 대화에 있어 상황에 따른 controller들의 인지 상태를 formal하게 정의하는 것으로 시작한다. 그것들을 message_type과 message_type의 선조건(pre-conditions), 후조건 (post- conditions) 그리고 만족조건(satisfying conditions)들을 기술하는데 이용한다. 궁극적으로는 그러한 인지 상태와 message_type의 사용을 연계 시키는 것을 기본 구조로 한다. Controller들의 인지 상태를 정의함에 있어서 Bel, Know, Want, Int라는 operator를 사용하는데, 이는 believe, know, want, intend라는 사람의 심리적 상태 (psychological state)들을 controller에게 적용하여 모델링 한다 (Singh, 1993). 그리고 controller 행동을 표현하기 위해서 Proc, SendMSG, Receive, Ans 등의 operator를 사용한다. 이와 같은 모델링은 heterarchical SFCS에서 각 controller들은 마치 사람처럼 자율적으로 의사 결정을 할 수 있는 능력을 가지고 있다는 데에 근거를 두고 있다. 이러한 operator들을 이용하여 일반적인 ICPL message_type들의 semantics는 다음과 같은 여섯 가지의 요소들로 기술될 수 있다(Labrou, 1996):

(1) message_type에 대한 자연어로서의 기술;
(2) illocutionary act를 기술하는 데 이용되는 formal expression;
(3) 선조건 - 임의의 message_type이 보내어질 수 있는 sender와 receiver의 조건;
(4) 후조건 - 임의의 message_type이 보내어진 후 sender와 receiver의 상태;
(5) 완성 상태(completion condition) - 대화 상황을 시작하는 message_type에 의해 제안되는 의도가 만족되었을 때, 이 message_type sender의 상태;
(6) 설명(comments) - message_type의 이해를 위해 필요한 설명.

3.3 Structure of ICPL

실제 두 controller간에 오고 가는 ICPL 메시지는 content가 메시지 wrapper에 의해 포장되어 메시지 expression으로 되고, 이것이 다시 통신 wrapper에 의해 포장된 것으로 이해할 수 있다. 실제로 전달하고자 하는 content 부분을 독립적이면서도 충분한 표현력을 지니기 위해서 메시지 유형별로 분류를 하고, network 환경에 독립적이기 위해서 통신 layer를 분리하는 것이 필요하다. 이러한 특성을 가지기 위해 ICPL은 layered 언어로

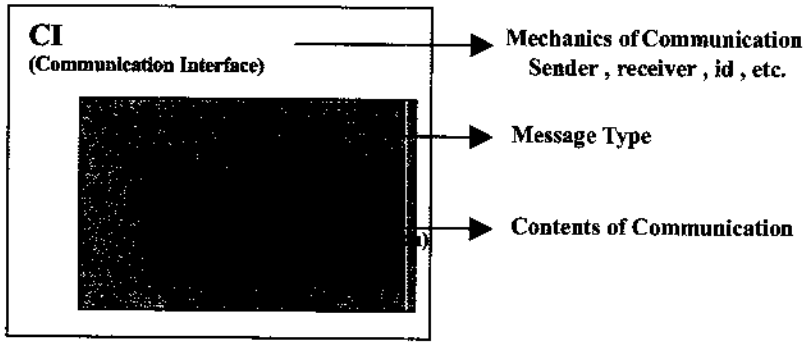

그림 2. ICPL의 3 layers.

디자인 된 것이다. 즉, 기존의 통신 프로토콜의 layer 구조처럼 ICPL도 <그림 2>에서 보는 바와 같이 IPC(Inter-Process Communication) layer, ICP (Integrated Communication Protocol) layer, CI (Communication Interface) layer의 3단계 layer로 나누어 디자인한다. IPC layer는 content layer로서 전달될 지식 그 자체를 임의의 언어를 통해 표현한 부분이다. ICP layer에서는 메시지 layer로서 이외의 부가적인 정보를 포함시킨다. 여기에는 content에 대한 정보와 이와 관련된 controller들의 speech act 종류 등이 있을 수 있다. 마지막으로 CI layer에서는 통신 layer로서 그 밖의 하위 레벨 통신을 위한 인자들이 추가된다. 여기에는 sender, receiver, 통신 모드 등의 관련된 정보가 있을 수 있다.

IPC Layer

ICPL을 사용할 때 controller는 content 부분을 자신의 명령어로서 구성한 후, 이를 ICPL 메시지 안에 싸서 보내게 된다. Content는 어떤 표현 언어를 사용하든지, ASCII string이든지 또는 많은 binary notation들 중의 하나이든지에 상관이 없지만 다만 machine interpretable은 해야 한다. 이에는 어떤 특정 정보의 선언이나 절차를 나타내는 지식도 포함될 수 있다. ICPL 구현은 메시지가 언제 시작되고 언제 끝나는가 외에는 메시지의 content 부분에 관심을 가지지 않는다. 따라서 실제 구현함에 있어서는 information architecture에서 제시된 Command /Respond 메시지를 확장하여 사용한다.

ICP Layer

IPC layer에서 기술된 내용에 부가적인 정보를 포장하여 메시지를 생성해 내는 layer이며 ICPL의 중심이 되는 부분이기도 하다. Speech Act Theory와 밀접한 관계를 가지는 layer라 할 수 있는데, 추가되는 부가적인 정보는 주로 content가 어떠한 종류의 speech act (request, query, response, error 메시지 등)를 띄고 전달되는가에 대한 것이다. 여기서 생성되는 메시지는 그 성격에 따라 (1) query, request 등과 같은 직접적인 정보 전달에 관련된 content 메시지와 (2) 직접적인 정보 전달은 아니나 이러한 활동에 관련된 정보에 대한 declaration 메시지로 나누어진다. 특히 declaration 메시지는 시스템 내에서 자신의 존재를 그 기능성과 함께 선언하거나 자신이 받고자 하는 어떠한 서비스에 대해 등록을 하거나 하는 등의 중요한 부분에 이용된다. 예를 들어, 새로운 controller가 SFCS에 등록될 경우 또는 고장이 나서 SFCS에서 등록이 취소될 경우 등에 declaration 메시지가 사용되어질 수 있다.

CI Layer

CI layer는 ICPL에서 가장 바깥 layer로서 위에서 언급된 IPC layer와 ICP layer를 거친 메시지를 통신에 관련된 부가 정보를 추가하여 통신을 위한 package의 형태로 만든다. 여기에서 추가되는 정보는 실제적인 통신 layer에서 사용되어질 정보인데 이는 sender, receiver, 통신 모드 그리고 메시지 ID 등의 ICP layer에서 얻어진 정보를 이용하여 얻는다.

3.4 Specification of ICPL

위에서 살펴본 이론적 배경과 구조를 가지는 ICPL의 사양에 대해서 알아본다. ICPL 사양은 위에서 본 ICPL의 3 layers 중에서 가장 중요한 부분인 ICP layer를 중심으로 이루어진다. Heterarchical architecture를 바탕으로 하는 SFCS에서는 여러 controller들의 복합체로 구현되는데 이러한 controller들은 다른 controller들로부터의 메시지에 자율적으로 반응할 수 있는 능력을 가져야 한다. ICPL은 controller들간의 통신에 관련된 언어/프로토콜이기에 메시지 transport에 대한 모델을 필요로 한다. 따라서 transport 단계에 대한 추상화된 모델을 다음과 같이 제시한다:

(1) 통신 link는 메시지 지연을 가질 수 있다.
(2) Controller가 임의의 메시지를 수신했을 때 어디로부터 온 것인지를 알 수 있다.
(3) Controller가 임의의 메시지를 보낼 때 어디로 갈 것인가를 명시해야 한다.
(4) 임의의 한 통신 link에 있어서 전달되는 메시지의 순서는 반드시 보내어진 순서이어야 한다.

표 1. Reserved message_type (A,B: controller의 이름, M: 메시지)

message_type	의미
advertise	(A는 B에게) M을 받으면 그것을 처리할 수 있다.
answer	(A는 B에게) 앞서 query에 대한 답을 한다.
award	A는 best bid를 제공한 B를 받아들인다.
broker	A는 B가 주어진 M에 대해 답을 구해 주길 바란다.
error	(A는 B에게) M이 처리되고 있지 않다고 전달한다.
monitor	A는 B의 응답을 지속적으로 update하고 싶어한다.
query	(A는 B에게) M에 관련된 질문을 한다.
recommend	A는 M에 대한 답을 할 수 있는 Controller를 원한다.
register	A는 주어진 이름으로 Shop 내에 등록되기를 원한다.
request	(A가 B에게) M에 관련된 내용을 원한다.
sorry	(A는 B에게) M에 대해 반응할 수 없다고 한다.
tell	(A가 B에게) M에 대해 말한다.
unregister	A는 자신이 등록한 내용을 취소한다.

(5) 메시지 transport는 안정적이어야 한다.

이러한 모델은 실제로 다양한 방법으로 그 구현이 가능하다. 예를 들어, Internet 상의 TCP/IP link일 수도 있고 Ethernet상에서 구축된 LAN상의 network link일 수도 있다. 어떤 경우든 위의 추상화된 모델만 만족한다면 ICPL은 제대로 적용되어 사용되어질 수 있으며 이러한 특징으로 인해 다양한 network 환경하에 쉽게 적용이 가능한 것이다.

앞서 알아본 syntax에서 message_type들은 message_type name과 parameter list로 실체화된다. 이용되는 형식은 다음과 같은 형태이다.

```
message_type_name
  : parameter_name parameter_value
  : parameter_name parameter_value
  .....
```

본 연구에서 정의된 message_type은 <표 1> 에서 보는 것과 같다.

ICPL의 message_type들은 그 semantics에 따라 (1) 기본 정보처리 - tell; (2) 요구 - request, query, broker, monitor; (3) 반응 - answer, error, sorry; (4) 협의 - recommend, award, advertise; (5) Network 관련 - register, unregister와 같은 범주로 나눌 수 있다.

3.5 Semantics of ICPL

그 의미에 따라 여러 범주로 나누었던 message_type들을 앞서 알아본 semantics에 기반한 형식에 따라 예를 들어보면 다음과 같다.

Request(A, B, M)

① A가 B에게 M에 관련되어 어떤 작업이나 정보를 요청한다.

② Want(A,Know(B,Proc(M)))

③ Pre(A) - Want(A,Know(B,Proc(M)))
Pre(B) - None

④ Post(A) - Know(A,Bel(B,CanProc(M)))
Post(B) - Know(B,Want(A,Know(B,Proc(M))))

⑤ Completion - Know(A,Bel(B,Proc(M)))

⑥ 단순한 명령 형태로 전해질 수도 있고 협의를 위한 정보를 요구하는 데 사용되어질 수도 있다. 이 메시지를 받은 controller는 반드시 TELL이나 SORRY, ERROR 같은 메시지로 반응해야 한다.

Query(A, B, M)

① A는 B의 능력 내에 M에 대해 반응을 원한다.

② Want(A,Know(A,Ans(M)))
③ Pre(A) - Want(A,Know(A,Ans(M)))
Pre(B) - None
④ Post(A) - Int(A,Know(A,Ans(M)))
Post(B) - Know(B,Want(A,Ans(M)))
⑤ Completion - Know(A,Ans(M))
⑥ 특정 메시지에 대해 질의를 하는 데 사용되어진다. REQUEST 메시지와 마찬가지로 ANSWER나 ERROR, SORRY 같은 메시지로 반응해야 한다.

Monitor(A, B, M)
① A는 B의 응답을 지속적으로 Update하고 싶어한다.
② Want(A,SendMSG(B,A,Ans(M)))
③ Pre(A) - Want(A,SendMSG(B,A,Ans(M)))
Pre(B) - Int(B,Proc(M))
④ Post(A) - Know(A,SendMSG(B,A,Ans(M)))
Post(B) - Know(B,Proc(B,M))
⑤ Completion - Know(A,Proc(B,M))
⑥ 계속해서 통신을 하며 메시지를 받을 때 사용되어진다. 이 메시지를 받은 controller는 자신의 status를 정해진 규칙에 따라 계속해서 TELL 메시지를 사용하여 전달해야 한다.

Broker(A, B, M)
① A는 B가 주어진 M에 대한 답을 구해 주길 원한다.
② Want(A,SendMSG(B,D,M))
Where D is a controller such that CanProc(D,M)
③ Pre(A) - Want(A,SendMSG(B,D,M))
Pre(B) - can be a facilitator
④ Post(A) - Know(A,SendMSG(B,D,M)
Post(B) - SendMSG(B,D,M)
⑤ Completion - SendMSG(B,A,Ans(M))
⑥ 직접적인 반응을 해주지 못할 때 이 메시지를 받은 controller가 대신 반응을 해줄 수 있는 controller에게 전해준다. 이때, 이 controller는 특정 설비의 controller가 아니고 shop 전체의 정보를 관리하는 resource manager일 수 있다.

Recommend(A, B, M)
① A는 자신의 메시지를 처리할 수 있는 controller를 원한다.
② Want(A,Bel(D,M))
Where D is a controller such that CanProc(D,M)
③ Pre(A) - Want(A,Bel(D,M))
Pre(B) - all controllers which are connected to A
④ Post(A) - Know(A,CanProc(B,M))
Post(B) - Int(B,Proc(M))
⑤ Completion - Know(A,Int(B,M))
⑥ 협의를 위한 Task_announcement에 사용되어진다. 연결된 여러 controller에게 수행해야 할 task를 알릴 때 사용되어질 수 있다.

Advertise(A, B, M)
① A는 B에게 메시지 M을 처리할 수 있다고 전한다.
② Int(A,Proc(A,M))
③ Pre(A) - Int(a,Proc(A,M))
Pre(B) - NONE
④ Post(A) - Know(A,Know(B,Int(A,Proc(A,M))))
Post(B) - Know(B,Int(A,Proc(A,M)))
⑤ Completion - Know(B,Int(A,Proc(A,M)))
⑥ RECOMMEND와 더불어 task_announcement에 따른 자신의 능력을 전달하고자 할 때 사용되어진다. 즉, Eligibility announcement에 사용되어질 수도 있고 RECOMMEND 메시지를 받고 자신의 능력을 표시하는 bid 메시지로 사용되어질 수 있다.

Award(A, B, ID)
① A는 B와 ID의 메시지에 대해 contract에 들어간다.
② Int(A,Know(B,Proc(ID)))
③ Pre(A) - Want(A,Know(B,Proc(ID)))
Pre(B) - Int(B,Proc(ID))
④ Post(A) - Know(A,Know(B,Proc(ID)))
Post(B) - Know(B,Proc(ID))
⑤ Completion - Know(B,Proc(ID))
⑥ 앞서 task_announcement와 그에 따른 bid들을 받아들여 그 중 best bid를 선정하여 그 controller와 contract를 할 때 사용되어진다.

4. 통신 메시지의 예

좀더 자세한 이해를 돕기 위하여 실제 통신 메시지의 예를 들어본다. Heterarchical SFCS에서 해결해야 할 문제들은 수없이 많이 있으며, 아직 문제 자체도 완전히 정의되어 있지 않다. 그러나 알려져 있는 여러 문제 중 통신과 information architecture에 결부되어 있는 몇 가지에 대해서 알아보면 다음과 같다(Duffie, 1990; Duffie and Piper, 1986):

(1) 부품 정보는 어디에 저장되며 또한 어떻게 관리되는가?
(2) 부품 가공에 대한 정보, 즉 공정 계획에 관한 정보는 어떻게 관리되는가?
(3) 생산 자원(머신, 공구 등)은 어떻게 관리되는가?
(4) 시스템의 교착 상태(deadlock)는 어떻게 피할 것이며, 어떻게 해결할 것인가? (Deadlock Detection and Resolution)
(5) 시스템에 어떻게 Fault - Tolerance를 부여할 것인가? (Fault Detection and Recovery)

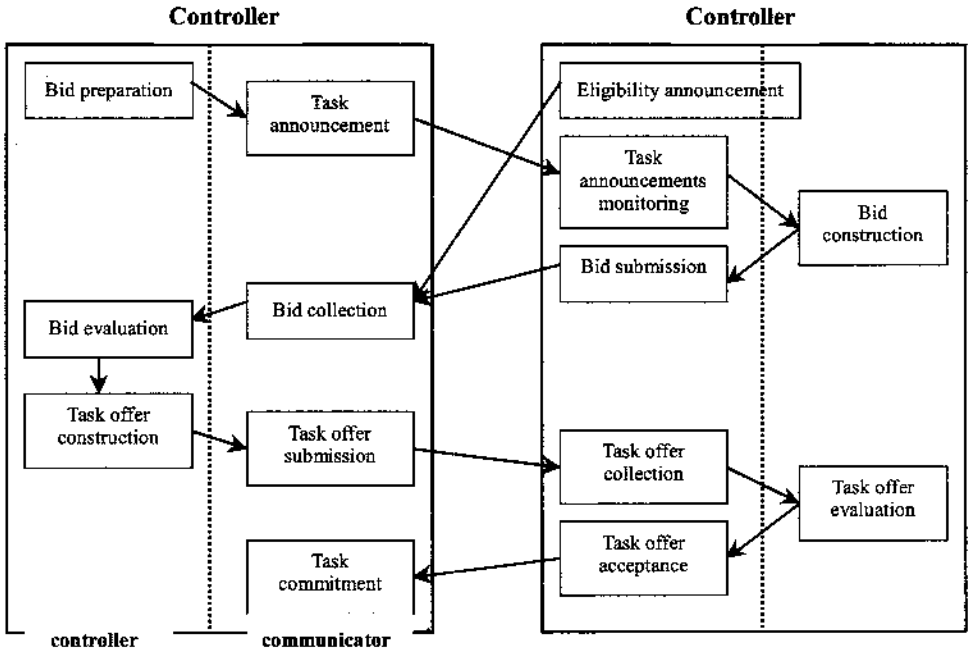

그림 3. 협상 과정.

(6) 어떤 task를 어느 resource에 할당할 것인가? (Task Allocation & Resource Allocation)

4.1 문제의 정의

앞서 언급된 문제 중에 여섯 번째 문제인 Task Allocation & Resource Allocation 의 경우를 예로 들어 보면, 이는 다시 다음의 세 가지 task로 분류되어질 수 있다:

(1) 가공 가능한 part가 여러 개 존재할 때, 어느 part를 가공할 것인가?

(2) 여러 resource 대안이 존재할 때, 어느 resource를 선택할 것인가?

(3) 위의 두 가지 문제가 복합되어 있을 때, 어떤 part를 어느 resource에 할당할 것인가?

따라서 part와 resource 간의 협상 과정을 실제 controller 간에 주고 받는 ICPL을 기반으로 한 메시지를 통해 살펴본다.

4.2 Part - Resource 협상

Heterarchical 제어 모델에서는 능동적으로 controller들간의 실시간 협상에 의해 planning, scheduling, execution의 기능을 수행해 나간다. 협상의 주체는 controller들이 되며 협상 process는 contract net 프로토콜(Lin and Solberg, 1992)을 바탕으로 하여 이루어진다. <그림 3>에 나타난 협상 과정은 작업을 알리는 task announcement, 그에 해당하는 bid들 중 best bid를 제시하는 controller와의 contract가 이루어지는 과정을 나타낸다(David and Smith, 1983; Smith, 1980). Bid는 작업을 수행하는 데 걸리는 시간, 비용 등으로 contract 결정의 기준이 되는데 시간은 due date 정보를 포함하여 task를 수행하는 데 소요되는 시간을 의미한다(Son, *et al*, 1995). 비용은 task를 수행함에 있어서 가공하는 데 필요한 tool change, tolerance 정보를 포함하여 산출해 낸다. 이와 같은 협상 과정에서 실제로 controller들간에 주고받는 메시지들은 ICPL을 기반으로 하여 생성된 것이다. Controller들이 직접적으로 처리하게 될 메시지들은 ICPL의 3 layer에서 IPC layer의 content 부분인데 이는 information architecture에서 제시된 command/respond 메시지를 기반으로 한다(Lee and Cho). 그러나 본 연구에서는 heterarchical architecture를 기반으로 하기 때문에 앞서 hierarchical architecture를 기반으로 연구된 메시지들만으로는 부족한 면이 많다. 즉, hierarchical architecture에서 중요시되지 않았던 자율적인 협상 부분을 담당해야 할 메시지들이 추가로 필요하게 된 것이다. <표 2>와 <표 3>은 이를 위해 확장된 메시지 specification을 나타낸다.

4.3 ICPL-based 통신 메시지

위에서 알아본 메시지와 <그림 3> 에서 보는 것과 같은 협상 과정을 바탕으로 실제 part - resource 협상을 위한 ICPL-based 통신 메시지들은 다음과 같다.

Task announcement

표 2. 확장된 메시지의 specification

Message	Parameter	Desscription
arrive	Pid, wfgid, Lid	Pid with wfgid arrived at Lid
cload	Pid, Lid	Pid must be moved from Lid
cunload	Pid, Lid	Pid must be moved to Lid
eplan	Pid	Pid must be planned
laod	Pid, Lid	needs to load Pid to Lid
machine	Pid, efgid	Pid with efgid must be machined
move	Pid, L1, L2	must pick a part from L1 to L2
pick	Lid	must pick a part from Lid
put	Lid	must put a part from Lid
replan	Pid	Pid must be replanned
s load	Pid, Lid	Pid must be moved from Lid
s unload	Pid, Lid	Pid must be moved to Lid
start	NCid, Oid	Ncid of Old must be downloaded
unload	Pid, Lid	needs to unload Pid from Lid
wplan	Pid, wfgid	wgfid of Pid must be planned
task()	Ttype, Pinfo	task has Ttype with Pinfo
bid()	Ttype, Ttime, Tcost	information with Ttype, Ttime, and Tcost
CanProcTask()	Ttype	Ttype can be processed
award()	Ttype, Pinfo	award task of Ttype with Pinfo

표 3. 메시지에 의해서 전달되는 parameters

Parameters	Definition
Pid	part identifier
L1, L2, Lid	location identifier
wfgid	workstation manufacturing feature graph iedtifier
efgid	equipment manufacturing feature graph idetifier
Eid	equipment sequence graph idetifier
Oid	operation sequence graph identifier
NCid	NC instruction identifier
Ttype	type of task
Ttime	processing time including information of due date
Tcost	processing cost including to lerance, tool change, etc.
Pinfo	information of part

PART controller가 Pinfo를 가지고 있는 Ttype의 task를 모든 controller들(parameter 값은 '*' 로 표현되어진다)에게 전달함으로써 협상 과정을 시작한다. 앞서 살펴본 ICPL의 semantics에 의해서 RECOMMEND는 ADVERTISE 메시지 형식으로 다른 controller로부터 메시지를 받을 것이다.

Eligibility announcement

M1 controller가 PART agent에게 자신이 Ttype을 가지고 있는 task를 수행할 수 있다고 통보한다. 실제로 M1 controller가 PART agent에게 이 메시지를 보내야 할지는 controller의 의사결정에 달려 있다. 이와 같은 통신 메시지는 하나의 통보 형식이므로 협상 과정과 무관하게 어느 때이든지 일어날 수 있다.

Bid submission

앞서 받은 task announcement 메시지에 대한 bid 메시지로서 Ttype을 가진 task를 Ttime과 Tcost로 수행할 수 있다고 전달한

다. 앞서 수신한 RECOMMEND에 대응하는 메시지로써 controller의 능력을 표현한다. 이에 대한 메시지를 보낼 수 없을 경우에는 SORRY 메시지를 대신 보낼 수 있다.

```
ADVERTISE
        :SENDER M1
        :RECEIVER PART
        :ID M001
        :CM TCP/IP
        :CONTENT
                CanProcTask(Ttype)
```

그림 4. Eligibility announcement.

```
RECOMMEND
        :SENDER PART
        :RECEIVER *
        :ID P001
        :CM TCP/IP
        :CONTENT
                task(Ttype, Pinfo)
```

그림 5. Task announcement.

```
ADVERTISE
        :SENDER M1
        :RECEIVER PART
        :ID M002
        :CM TCP/IP
        :CONTENT
                bid(Ttype, Ttime, Tcost)
```

그림 6. Bid submission.

```
AWARD
        :SENDER PART
        :RECEIVER M1
        :ID P002
        :CM TCP/IP
        :CONTENT
                award(Ttype, Pinfo)
```

그림 7. Task offer submission(Award).

Task offer submission(Award)

PART agent는 M1 controller과 contract를 이루고 Ttype과 Pinfo를 가진 task에 대해 수행 명령을 내린다.

Reporting result and termination

이후 PART agent는 M1 controller로부터 monitoring 정보를 받게 되고 task가 수행되었을 경우 termination 메시지를 전달함으로써 모든 협상 과정이 끝나게 된다.

5. 적용 및 결과

본 연구에서 구현하고자 하는 통신은 <그림 8>과 같이 구성된 여러 device들을 제어하는 heterarchical SFCS의 구성원인 controller들이 담당한다. CNS(Controller Name Server)는 controller의 이름만을 가지고 그 물리적인 주소를 알아내는 데 사용되어지고 Facilitator는 controller들간의 상호 메시지 전달에 사용되어진다. 각 controller에 구현된 communicator는 Windows NT 4.0을 운영 체제로 한 IBM PC 586급 호환 기종인 PC에서 구현되었다.

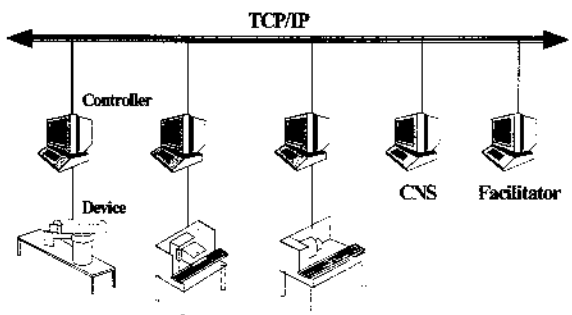

그림 8. Controller들과의 통신망 구성.

ICPL의 효용성을 검증하기 위하여 본 연구에서 구현한 프로그램은 외부와의 통신을 맡고 있는 CI와 메시지 해석을 맡고 있는 ICM(ICPL Conversation Module)이다. 이 프로그램은 ICPL을 기반으로 하는 메시지들을 처리할 수 있고 파일 송수신을 위한 FTP(File Transfer Protocol)를 포함하고 있다. 실제 메시지와 파일을 주고받는 과정은 <그림 9> 에서 보는 것과 같다.

Facilitator는 메시지를 구성하는 것이 주요 기능이지만 controller의 어떠한 의사 결정 과정에도 관여하지 않으며 그 메시지의 전달 과정에만 관여를 한다. 또한 CNS의 기능과 FTP Server와의 전달을 위한 매개체 역할도 한다. Facilitator가 없다면 각 controller는 다른 controller들의 고유한 이름과 물리적인 주소들의 최신 정보들을 모니터링해야 한다. 이는 CNS의 기능과 중복되는 것이므로 그 기능을 CNS로 한정하고 메시지 전달의 효과적인 구현을 위하여 facilitator가 존재하게 된다.

본 연구에서 구현한 프로그램으로 part - resource 협상 과정을 보면, 우선 <그림 10> 과 <그림 11>과 같이 facilitator를 setup 한다. Setup은 facilitator가 위치하고 있는 host의 물리적인 주소 입력과 FTP 서비스를 위한 계정과 비밀 번호 입력으로 이루어져 있다.

<그림 12> 에서 보는 것과 같이 CI는 전체적으로 4개의 부

분으로 나누어져 있다:

(1) Status : 등록된 controller의 이름과 connect/disconnect 할 수 있는 기능을 갖고 있다. 또한 CNS의 기능을 볼 수 있다.

(2) ICPL : 실제 처리할 메시지들을 구성하는 부분으로 receiver와 content를 작성하여 메시지를 전송한다.

(3) Register : Facilitator와 CNS에 자신의 이름을 등록하는 곳이다.

(4) FTP : 파일 송수신을 위한 setup이다.

(5) SFCS에 소속된 controller는 Register를 통해서 자신의 이름을 등록하게 되고 Status에서 전체 시스템과 연결되어 ICPL을 통해서 실제로 메시지를 주고받게 되는 것이다.

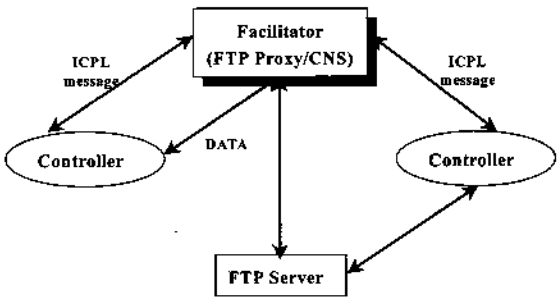

그림 9. Controller들간의 데이터와 메시지 전달 구조.

<그림 13>과 <그림 14>에는 4절의 통신 예를 수행하였을 때의 각 controller의 CI를 보여 주고 있다. <그림 13>은 machine1 controller(M1_C)의 CI를 보여주는데 오른쪽 창은 수신된 메시지를 나타내고 있다. 처음 PART_C로부터 task announcement (RECOMMEND)를 받고 최종적으로 AWARD 메시지를 받는 것을 보여 준다. 또한 <그림 14>는 part controller(PART_C)의 CI를 보여 주는데, 자신이 보낸 task announcement에 대한 각 controller들의 bid (ADVERTISE) 메시지를 받고 그 중 best bid에 해당하는 M1_C에게 AWARD 메시지를 주고 있음을 알 수 있다.

6. 결론 및 추후 연구

기존의 SFCS에서controller들은 hierarchical architecture하에 수동적인 작업을 해왔으며 서로간에 command/respond 메시지만을 주고받았었다. 그러나 heterarchical architecture로 발전해 가면서 능동적이고 자율적인 능력을 가지게 되었고 이에 따라 다른 controller들과 협상과 통신을 통해서 자율적으로 문제를 해결해 나가게 되었다. 따라서 controller들은 자율적인 협상과 통신이 중요시됨으로써 기존의 메시지뿐만 아니라 file data를 비롯한 협상을 위한 충분한 표현력을 가진 통신 능력을 필요로 하게 되었다.

본 연구에서는 autonomous agent를 기반으로 한 heterarchical SFCS의 구현에 있어서 가장 큰 걸림돌인 통신 문제를 해결하고자 표준화된 통신 언어/프로토콜인 ICPL(Integrated Communication Protocol and Language)을 제안하였고 디자인하였다. 또한 그 이론적인 배경과 사양에 대해서도 설명하였다. 추후 연구과제로는 현재 진행중인 ICPL을 기반으로 한 독립적인 communication architecture를 개발한다면, heterarchical SFCS에 적용되어 자율적이고 협상 중심의 통신 문제를 해결해 줄 수 있을 것이다.

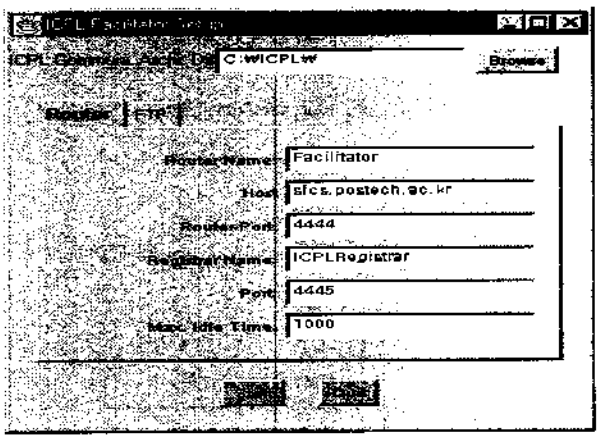

그림 10. Facilitator의 setup.

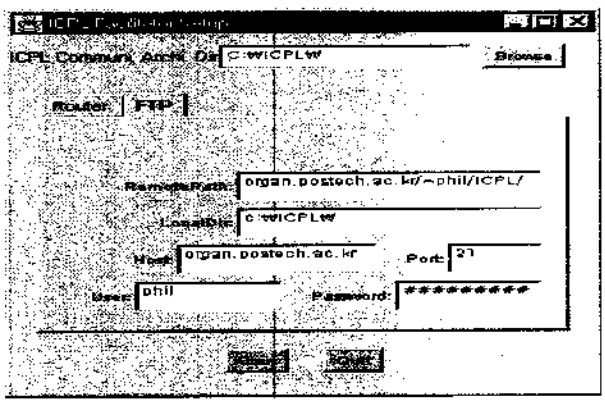

그림 11. FTP 서비스를 위한 setup.

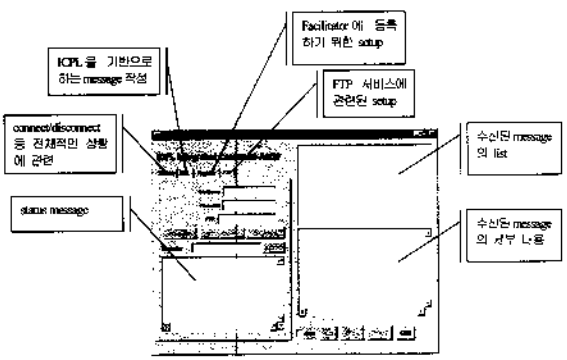

그림 12. Communication Interface의 전체 모습.

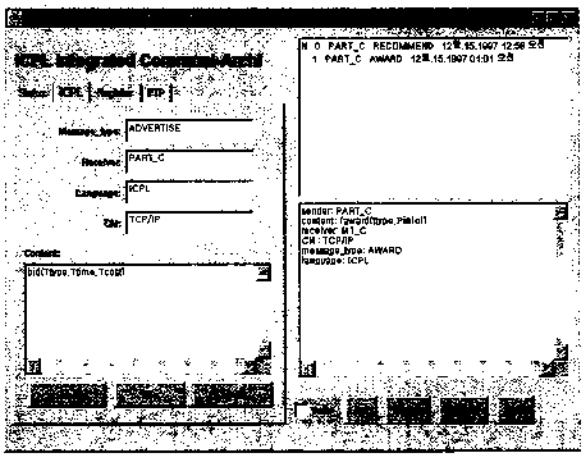

그림 13. M1_C의 CI.

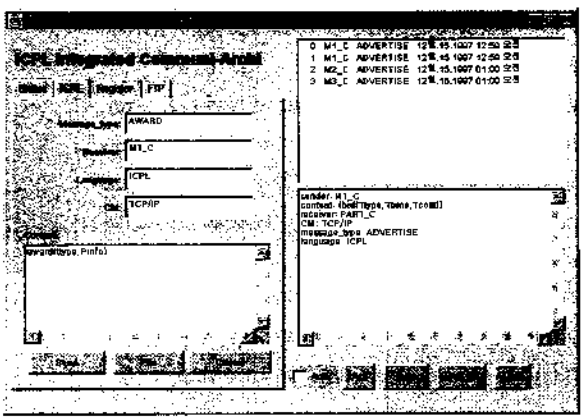

그림 14. PART_C의 CI.

감사의 글

본 논문이 보다 나은 내용과 표현을 갖도록 도와 주신 두 분의 심사위원들께 감사의 말씀을 드립니다

참고문헌

ARPA Knowledge Sharing Initiative (1993), Specification of the KQML agent communication language, *ARPA Knowledge Sharing Initiative, External Interfaces Working Group working paper.*

David, R. and Smith, R. G. (1983), Negotiation as a metaphor for distributed problem solver, *Artificial Intelligence*, 20(1), 63-109.

Dilts, D. M., Boyd, M. P. and Whorms, H. H. (1991), The evolution of Control Architectures for Automated Manufacturing Systems, *Journal of Manufacturing Systems*, 10(1), 79-93.

Duffie, N. A. (1990), Synthesis of heterarchical manufacturing systems, *Computer in Industry*, 14, 167-174.

Duffie, N. A. and Piper, R. S. (1986), Nonhierarchical Control of Manufacturing Systems, *Journal of Manufacturing Systems*, 5(2), 137-139.

Finin, T., McKay, D., Fritzson, R. and the KQML Advisory Group (1992), *An overview of KQML: A Knowledge Query and Manipulation Language.*

Hong, S., Cho, H. and Jung, M. (1995), A Petri-Net based Execution Model of Processing Equipment for CSCW-based Shop Floor Control in Agile Manufacturing, '95 *춘계공동학술대회* 논문집, 한국경영과학회/대한산업공학회, 208-215.

Kim, H., Cho, H. and Jung, M. (1995), An intelligent Planner of Processing equipment for CSCW-based Shop Floor Control in Agile Manufacturing, '95 *춘계공동학술대회논문집*, 한국경영과학회/대한산업공학회, 185-192.

Kramer, T. R. and Senehi, M. K. (1993), Feasibility Study: Reference Architecture for Machine Control Systems, NISTIR 5279, *National Institute of Standards and Technology*, MD, USA.

Labrou, Y. (1996), Semantics for an Agent Communication Language, Ph.D. Dissertation, University of Maryland, Baltimore, Maryland.

Lee, I. and Cho, H., Specification of Information and Message Requirements for Shop Floor Control through Function Modeling, *accepted for publication in International Journal of Computer Integrated Manufacturing.*

Lin, G. Y. and Solberg, J. J. (1992), Integrated Shop Floor Control using Autonomous Agents, *IIE Transactions*, 24(3), 57-71.

Maturana, F. P. and Norrie, D. H. (1997), Distributed decision-making using the contract net within a mediator architecture, *Decision Support Systems*, 20, 53-64.

Messina, G. and Tricomi, G. (1990), Manufacturing Communication Architectures, *Computers in Industry*, 13, 285-293.

Searle, J. R. (1969), *Speech Acts*, Cambridge University Press, Cambridge, UK.

Shoham, Y. (1993), *Agent-Oriented Programming, Artificial Intelligence*, 60, 51-92.

Singh, M. P. (1993), Semantics for Speech Acts, *Annals of Mathematics and Artificial Intelligence*, 8, 47-71.

Smith, R. G. (1980), The contract net protocol: High-level communication and control in a distributed problem solver, *IEEE Transactions on System, Man, Cybernetics*, 29(12), 1104-1113.

Son, K., Cho, H. and Jung, M. (1995), Assembly System Control under CSCW-based Shop Floor Control, '95 *춘계공동학술대회* 논문집, 한국경영과학회/대한산업공학회, 201-207.

Tiemersma, J. J., Curtis, W. and Kals, H. J. J. (1993), Communication and Control in Small Batch Part Manufacturing, *Robotics and Computer-Integrated Manufacturing*, 10, 123-129.

Valenzano, A., Demartini, C. and Ciminiera, L. (1992), *MAP and TOP: Communications Standards and Applications*, Addison-wesley.