재구성 용이한 생산시스템을 위한 프로세스 엔진의 개발
Development of Process Engine for Reconfigurable Manufacturing System (RMS)

*김진회, 이원균 , #민병권, 이상조
*J. H. Kim, W. K. Lee , #B.- K. Min (bkmin@yonsei.ac.kr) , S. J. Lee
연세대학교 기계공학과

Key words : manufacturing system, factory control, process control

1. 서론

재구성 용이한 생산시스템 (RMS)을 위한 제어기 개발에서 RMS 의 재구성 용이성을 높이기 위해서 다양한 컨트롤러 에이전트들로 이루어진 멀티 에이전트 시스템 구조를 사용했다. 멀티 에이전트 시스템가 가지고 있는 문제 중 하나가 에이전트 간의 상호작용 문제이다. 에이전트는 부분적으로 자신이 담당하는 지역적인 문제 을 해결하기 위한 프로그램이지만, 전체적으로는 에이전트들이 속한 시스템의 목표를 해결하는 것이 더욱 중요하다. 그래서, 상호작용 문제가 발생하는 것이다. 상호작용 문제는 크게 Coordination, Cooperation, Collaboration 으로 나누어 볼 수 있다. Coordination 는 에이전트 간의 지역적인 기능이 서로 상충되는 경우에 최적의 효과를 얻기 위해서 지역적인 기능을 조정하는 것을 말한다. Collaboration 은 하나의 공통 목표를 달성하기 위하여 에이전트들이 협력하는 경우를 말하며, Cooperation 은 에이전트가 다른 에이전트의 목표를 달성하기 위하여 협조하는 경우를 말한다.

에이전트 간의 상호작용을 해결하는 방법은 크게 두 가지로 나눌 수 있다. 시스템에 에이전트들의 정보와 전체 시스템의 정보를 블랙보드라는 중앙의 메모리 공간에 저장하여 이 정보를 통하여 시스템의 중심에서 상호작용를 해결하는 방법과 상호작용이 발생할 경우, 에이전트들이 자율적으로 통신하여 해결하는 방법으로 나눌 수 있다. 각각의 방법은 장단점이 있다. 첫 번째 블랙보드를 이용하는 방법은 제어의 일관성을 보장할 수 있고, 상대적으로 단순하다는 장점을 가지고 있지만, 시스템의 변화에 취약하여 새롭게 추가되는 모듈들의 상호작용을 해결하기 어렵다는 단점을 가지고 있다. 두 번째, 통신을 이용하는 방법은 시스템의 변화가 상대적으로 쉬워 유연하다는 장점이 있지만, 상호작용 문제해결을 위해서 지속적으로 통신을 해야 하기 때문에 통신로드가 발생할 수 있다는 단점이 있으며, 상호작용 문제을 해결하는 모듈이 각각의 에이전트에 있기 때문에 시스템이 복잡해질 수 있는 점도 단점이 된다.

위에서 제시한 방법 모두 본 시스템에 그대로 적용하기에는 무리가 있기 때문에 본 연구에서는 에이전트 구조를 이용해서 전체 시스템의 재구성 용이성을 저하시키지 않으면서, 에이전트 간의 상호작용 문제를 해결하는 방법을 찾아보도록 하겠다.

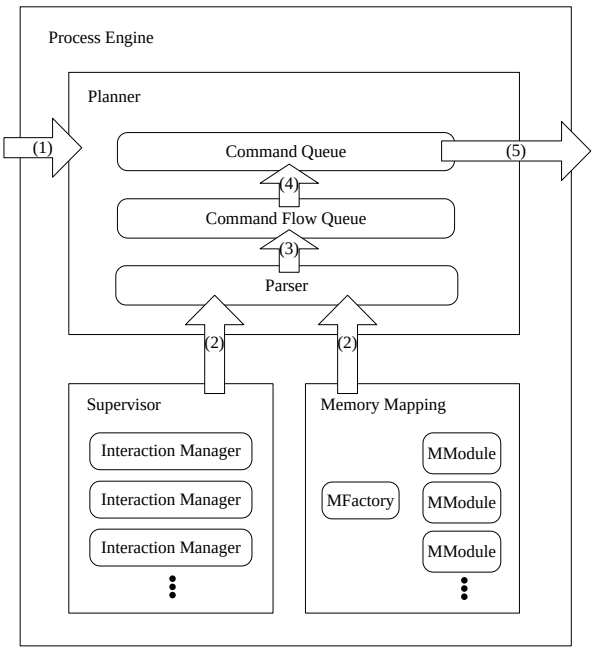
2. 프로세스 엔진의 메커니즘

본 시스템에서는 블랙보드 방법과 통신 방법의 장점을 모두 얻을 수 있는 방법을 찾아보았다. Fig.1 과 같이 XML 로 상호작용 문제과 그 해결 방법을 정의해놓고, 시스템 변경 필요 없이 정의된 XML 파일을 참고하여 상호작용 이 일어날 때마다 상호작용 해결 방법을 실행하게 된다. Fig.1 에 나와 있는 Interaction Problems Definition 에는 “Condition” 이라는 항목으로 상호작용 (Interaction)상황을 정의하고 있다. 전체 공정을 제어하는 소프트웨어인 RMS Executive 에서 프로세스 다이어그램에 따라서 개별 하드웨어 모듈을 제어하는 컨트롤러 에이전트에게 명령을 내리기 전, 명령 후 일어날 상황을 “Condition”에 따라서 판단하여 상호작용이 일어날 것으로 판단될 경우, 같이 정의 되어 있는 명령어 흐름을 먼저 실행하여 상호작용을 해결하고, 본래 실행하려 했던 명령어을 실행하게 된다. 이렇게 상호작용을 정의한 파일은 새로운 모듈이 추가되거나 삭제될 경우에 추가되거나 삭제될 상호작용을 XML 에 새롭게 정의하거나 삭제하면 상호작용 해결 모듈을 변경할 수 있다. 먼저 Interaction Action Definition 에 대해서 살펴보고, Process Engine 의 작동 매커니즘에 대해서 알아보도록 하겠다.

```
<Interaction>
<Name>Conflict</Name>
<Type>Coordination</Type>
<Condition>
((Module1.x EQ 10) AND (Module1.y LT 10))
</Condition>
<CommandFlow>
<Command>Module1.Move px=10 py=40</Command>
<Command>Module2.Move px=0 py=50</Command>
<Command>Module2.Move px=100 py=100</Command>
</CommandFlow>
</Interaction>
```

Fig. 1 Interaction Action Definition

Fig.1 을 살펴보면 여기서 정의된 상호작용의 이름은 *Conflict* 이며, 그 타입은 *Coordination* 이다. Fig.1 에서 보는 것처럼 RMS Executive 에 작성된 프로세스 다이어그램에 따라서 해당 컨트롤러 에이전트 에게 내리는 명령을 “Raw Command”라고 정의했다. Raw Command 가 실행됐을 경우 변경되는 팩토리의 정보는 Fig.2 에 시스템 내에 소프트웨어 모듈인 MFactory 와 MModule 에 맵핑 되어 있다. MFactory 는 팩토리에 대한 정보를 저장하고 있는 소프트웨어 모듈이며, MModule 은 팩토리를 구성하는 하드웨어 모듈의 정보를 저장하고 있는 소프트웨어 모듈이다. Fig.2 에서 상호작용 조건 하나당 하나씩 Supervisor 안에 Interaction Manager 가 존재한다. Interaction Manager 가 Condition 과 상호작용 해결을 위한 Command Flow 를 Fig.2 의 Planner 에게 넘겨주게 되고, 이것을 이용하여 Planner 는 내부에 있는 파서 (Parser)로 조건문을 판단하여 참일 경우, Command Flow 를 Command Flow Queue 에 저장하게 된다. 참고로 Fig.1 에 나온 ***(Module1.x EQ 10) AND (Module1.y LT 100)***의 의미는 Module1 의 X 가 10 과 같고 (EQ = Equals), Module1 의 y 가 100 보다 작을 경우 (LT=Less Than)를 Interaction 이라고 정의한 것이다. Command Queue 에서 하나씩 Command 를 빼내어 실행한다. 본 프로그램의 조건문에는 미리 정의된 논리연산자, 산술연산자, 비교연산자를 사용하여 상호작용을 정의할 수 있다. 위에 제시된 조건문이 참일 경우에는 Module1 을 x, y 의 위치를 (10,40)으로 이동시키고 (Module1.Move px=10 py=40), Module2 의 위치를 (0,50)으로 이동시킨다. (Module2.Move px=0 py=50) 그리고, 마지막으로 Module2 의 위치를 (100,100)으로 이동시키는 명령을 시킨 후에 원래 실행하려했던 명령어를 실행시킨다.



(1) Input *raw command*
(2) Return *interaction condition* and *command flow*
(3)Enqueue in *command flow queue* if the result of *condition* evaluated by *parser* is true
(4) Enqueue *command flow queue* to *command queue*
(5)Operate *augmented command* as *command queue*

Fig. 2 Internal flow for solving Interaction

1.Process Engine 으로 Process Diagram 의 Command 공급
2.Process Engine 에서 Command 를 처리하여 CommandFlow 를 만들고, CommandFlow 에서 하나씩 Command 실행
- A. Command를 Interaction Manager에서 check하여 CommandFlow 생성
- B. Interaction Manager에서는 Variable Heap에 저장된 Post Set (Action 이후의 상태 변수)와 현재 팩토리 상태를 이용하여 Interaction 체크
- C. Command의 Post Set을 체크하여 파라미터 값이나 실제 값인지 구분하여 저장 (- Variable Heap)
- D. Interaction이 발생할 경우, Command에 대한 Command Flow를 Command Flow Queue에 저장

Fig.3 Process Engine Mechanism

Fig.2 는 Process Engine 의 간략한 구조와 명령어처리흐름을 나타낸 그림이고, Fig.3 은 Process Engine 이 상호작용을 처리하는 매커니즘을 간략하게 설명한 그림이다. Fig.3 의 1 단계와 같이 RMS Executive 에서 작성한 프로세스 다이어그램을 실행하면서 프로세스 다이어그램을 구성하는 블록에 해당하는 하나의 Command 을 프로세스 엔진 에 Command 을 입력한다. Fig.3 의 2 단계와 같이 프로세스 엔진은 입력받은 Command 의 상호작용을 처리하여 Command Flow 를 만든다. Command Flow 를 만드는 과정은 미리 정의된 Interaction Condition 당 하나씩 담당하는 Interaction Manager 가 체크하여 상호작용을 해결되지 않은 Command Flow 를 만든다. 그 다음에 Interaction Manager 에서는 별도로 마련된 Variable Heap 이라는 저장공간에 PostSet (Command 실행 이후의 팩토리의 변화된 상태 변수)과 현재 팩토리 상태를 이용하여 상호작용을 검사한다. Command 의 PostSet 을 체크하여 파라미터 값이 실제 값을 구분하여 저장한다. 마지막으로 상호작용이 발생할 경우 이것을 해결하기 위한 Command 에 대한 Command Flow 를 Command Flow Queue 에 저장한다.

3. 결론

공정을 설계하고 실행하는 RMS Executive 소프트웨어와 개별 하드웨어 모듈을 제어하는 Controller Agent 를 이용하여 재구성 용이한 마이크로팩토리를 위한 제어기를 개발했다. 이 제어기는 하드웨어 모듈 간의 의존성을 낮추기 위해서 멀티 에이전트 구조를 사용했다. 독립적으로 작동하는 에이전트로 제어되는 하드웨어 모듈은 생산시스템의 재구성 용이성을 높였으나, 멀티 에이전트 구조는 독립적으로 작동하는 에이전트들이 같은 물리적, 논리적 환경에 있기 때문에 여러가지 상호작용 문제가 발생할 수 밖에 없다. 에이전트 간에 발생하는 상호작용을 해결하기 위해서 멀티 에이전트 시스템에서는 블랙보드 방식과 통신 방식을 사용하지만, 블랙보드 방식은 단순하고 견고한 처리 구조를 가진 반면에 처리할 수 있는 상호작용이 미리 지정되어야 있어야한다는 점에서 처리할 수 있는 상호작용의 종류가 한정되어있다는 단점을 가지고, 통신 방식은 다양한 상호작용을 처리할 수 있다는 장점을 가지고 있지만, 처리방식 자체가 복잡하다는 단점을 가진다. 이에 본 연구에서는 처리방식은 블랙보드 방식을 사용하되 다양한 상호작용을 처리할 수 있게 하기 위해서 별도의 상호작용 처리 스펙을 정해놓고 에이전트가 추가될 때마다 발생하는 상호작용을 정의하여 유연하게 추가할 수 있는 프로세스 엔진을 개발했다. 이 프로세스 엔진을 통해서 독립적으로 작동하는 에이전트를 이용하여 시스템의 재구성 용이성은 그대로 유지하면서 에이전트 간에 발생하는 상호작용을 유연하게 처리할 수 있도록 했다.

후 기

본 논문은 차세대 마이크로팩토리 시스템 개발 사업의 일환으로 지식경제부의 지원을 받아 수행되었으며, 이에 관계자 여러분께 감사드립니다.

참고문헌

1. ElMaraghy, H.A., *Flexible and reconfigurable manufacturing systems paradigms.* International Journal of Flexible Manufacturing Systems, 2005. 17(4): p. 261-276.
2. Okazaki, Y., N. Mishima, and K. Ashida, *Microfactory - Concept, history, and developments.* Journal of Manufacturing Science and Engineering-Transactions of the Asme, 2004. 126(4): p. 837-844.
3. Mehrabi, M.G., A.G. Ulsoy, and Y. Koren, *Reconfigurable manufacturing systems: Key to future manufacturing.* Journal of Intelligent Manufacturing, 2000. 11(4): p. 403-419.
4. 김진회, 민병권, 이상조, 재구성 용이한 마이크로팩토리 제어기 개발, 한국공작기계학회 2007 춘계학술대회 논문집
5. 김진회, 박성수, 윤희택, 이원균, 민병권, 이상조, 모듈형 생산시스템 제어를 위한 소프트웨어 개발, 한국정밀공학회 2007 년도 추계학술대회논문집