HLA/RTI 기반의 페트리 네트 분산 시뮬레이션 모델

임동순*

Distributed Simulation models of Petri Nets under HLA/RTI

Dong-Soon Yim

Abstract

In this study, a distributed simulation of Petri net models under HLA/RTI framework is considered. Throughout our modeling experiences, it is recognized that the proper use of interface specification and time management services are important in order to achieve successful implementation of RTI. The interfacing tokens between Petri net models are distinguished as information entity and physical entity. Both entities are modeled as **InteractionClass** in order to send and receive messages. For the synchronization of local simulation clocks, a conservative method with **NERA** service is considered. A cell manufacturing system is modeled and implemented with RTI to illustrate the distributed simulation of Petri net models.

Key Words: distributed simulation, Petri nets, HLA/RTI

* 한남대학교 산업시스템공학과

1. 서론

이산사건 시뮬레이션의 실행속도를 증가시키고, 거대한 모델의 메모리 문제를 해결하기 위한 방법으로 병렬 시뮬레이션 기법이 연구되어왔다. 기존의 방법인 순차적 시뮬레이션은 하나의 시뮬레이션 모델을 한 컴퓨터에서 시간에 따른 순차적 사건 순으로 실행시키는 것에 비하여 병렬시뮬레이션은 여러 대의 컴퓨터 또는 다중 프로세스를 갖는 한 컴퓨터에 시뮬레이션 모델을 분산시킴으로써 실행속도를 향상시키는데 목적이 있다[1,3].

병렬시뮬레이션에서 추구하는 시뮬레이션 모델의 분산은 모델링 관점에서도 많은 장점을 제공한다. 특히, 복잡한 시스템을 대상으로 하는 경우에 모델이 거대해져 전체 시스템을 하나의 모델로 묘사하는데 어려움이 존재한다. 만약, 모델링되는 시스템이 각각의 모듈화된 단위 모델로 쉽게 작성되고, 단위 모델간의 인터페이스가 쉽게 구현될 수 있다면 분산 모델링은 좋은 장점을 제공한다.

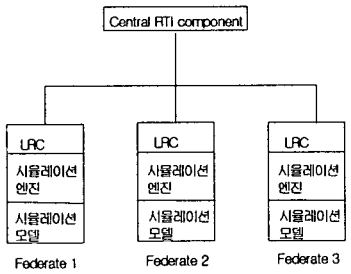
HLA(High Level Architecture)는 미 국방성의 주도하에 개발된 것으로 여러 단위 모델로부터 거대한 시뮬레이션 모델을 생성하기 위한 소프트웨어 구조이다[5]. HLA는 (1) 기본적인 원칙을 정의한 규칙, (2) 여러 시뮬레이터에 공통적으로 관심이 되는 정보를 묘사하기 위한 표준 포맷을 제공하는 모델 템플리트, (3) 시뮬레이터 간의 정보 교환을 위한 인터페이스 규격으로 구성되어 있다[2,5]. HLA 시뮬레이션은 RTI(Run Time Infrastructure)라는 소프트웨어를 통하여 연결된 여러 시뮬레이터를 포함하고 있다. 각 시뮬레이터들을 페더레이트(federate)라고 부르고, RTI를 통하여 연결된 페더레이트의 집합을 페더레이션(federation)이라고 부른다. RTI는 다수의 시뮬레이션 모델들을 연결시키기 위한 서비스를 제공하는 특별한 목적의 분산 운영체제라고 할 수 있다. HLA의 인터페이스 규격은 페더레이션 실행 중에 RTI 또는 각 시뮬레이션 모델이 수행하는 서비스의 집합을 정의하고 있다. 이러한 서비스의 집합은 페더레이션 관리, 선언관리, 객체관리, 소유권관리, 시간관리, 데이터 분배관리 등의 6가지로 구성되어 있다. RTI를 이용한 분산 시뮬레이션은 이러한 인터페이스 서비스를 이용하여 모델 간에 메시지를 주고 받음으로써 가능하다. RTI 소프트웨어는 <그림 1>과 같이 CRC (Central RTI Component)와 LRC (Local RTI Component)로 구성되어 있다. 각 페더레이트는 자신의 시뮬레이션 모델과 엔진을 갖고 있고, LRC를 통하여 RTI와 교신한다.

이산사건 시뮬레이션은 공통적으로 사건리스트의 처리를 필요로 한다. 미래에 발생할 사건을 미래사건 리스트에 발생시간 순으로 저장하여 이 중 가장 빠른 시각을 갖는 사건을 꺼내 시뮬레이션 시각을 이 시각으로 진전시키고, 해당 사건을 발생시킨다. 분산 모델을 고려할 경우 각 모델은 각자의 시뮬레이션 엔진에 의해 이와 같은 사건 리스트 처리를 한다. 그러나, 모델 간에 연결되는 사건을 처리하는 절차가 필요하다. 한 모델의 사건이 다른 모델의 사건을 유발한다면 이 사건은 두 모델 간의 인터페이스를 필요로 한다. 만약, 두 모델의 시뮬레이션 시각이 서로 다른 경우 이는 두 모델의 사건 발생시각을 동기화하여야 함을 의미한다.

병렬시뮬레이션에서 시뮬레이션 시각을 동기화하는 방법으로는 크게 두가지로 보수적 방법[3]과 낙관적 방법[4]이 있다. RTI 소프트웨어는 이 두가지 방법을 모두 지원하고 있다.

본 연구에서는 페트리 네트를 이용한 분산된 시뮬레이션 모델들을 RTI기반으로 통합하여 시뮬레이션을 수행한 결과를 제시한다. 특히, 시뮬레이션 모델간의 인터페이스 방법과 시뮬레이션 시각을 동기화하기 위한 보수적 방법의 RTI 서비스 이용 사례를 소개한다. 본 연구에서의 시뮬레이션 실행은 예측된 최소한의 다음 시각 간격인 lookahead 값이 0인 경

우를 허용하여야 한다. 이러한 상황에서 보수적 방법을 사용하기 위하여 모델링 측면에서 고려하여야 할 사항들을 논의한다.



<그림 1> RTI를 이용한 분산 시뮬레이션

2. 자바 기반의 페트리네트 시뮬레이션 도구

본 연구에서는 생산시스템에 적용될 수 있는 페트리네트의 모델링 및 시뮬레이션을 위해 자바 언어를 이용하여 시뮬레이션 도구를 작성하였다. 모델링 측면에서 시스템의 상태 또는 행위를 플레이스(place)로 나타내고, 사건을 트랜지션(transition)으로 묘사한다. 그러나, 행위 만을 하나의 플레이스로 표현할 경우 많은 플레이스를 야기하므로 하나의 행위와 이에 수반되는 행위 종료사건, 그리고, 대기상태를 하나의 플레이스로 모델링 가능토록 하였다. 또한, 시스템 내의 자원과 흐름 객체들은 토큰(token)으로 모델링 하였다. 사용된 페트리네트는 기본적으로 일반적인 페트리 네트에 속하나 생산시스템의 특성을 고려하고, 시뮬레이션 실행을 위해 다음과 같은 모델링 요소를 추가하였다.

1) 플레이스에 토큰이 도착하면 주어진 지연시간을 갖는다.
2) 플레이스에 있을 수 있는 최대 토큰의 수인 용량이 설정되어 있다.
3) 기본적인 토큰은 토큰 클래스의 인스턴스이다.
4) 트랜지션의 출력아크에는 이동되는 토큰의 집합이 정의된다.

한 토큰이 플레이스에 도착하면 주어진 지연시간을 갖는다. 지연 시간 후 토큰은 출력 트랜지션을 발사(fire)할 수 있는 활동상태가 된다. 각 플레이스에는 용량이 설정되어 용량 이상의 토큰 유입은 금지된다. 따라서, 트랜지션 t가 활성화될 수 있는 조건은 다음과 같다.

트랜지션 활성화 조건

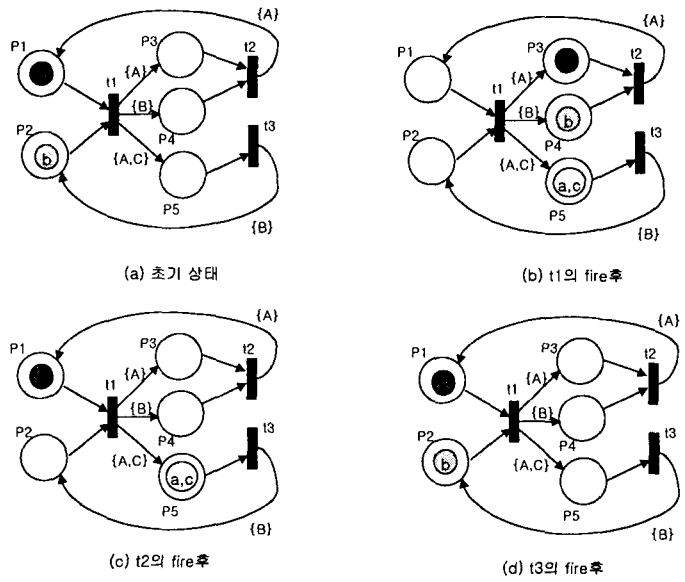
1) 트랜지션 t의 각 입력 플레이스에는 활동상태의 토큰이 하나 이상 존재한다.
2) 트랜지션 t의 각 출력 플레이스에 있는 토큰의 수는 용량보다 적다.

트랜지션 t가 활성화 조건을 만족하면 즉시 발사되어 다음의 결과를 가져온다.

트랜지션 발사규칙

1) 트랜지션 t의 입력 플레이스에 있는 활동상태의 토큰 중 하나를 선택하여 트랜지션으로 이동시킨다.
2) 트랜지션 t의 각 출력 아크에 정의된 토큰 집합에 따라 이동된 토큰들을 복사하거나, 새로운 토큰을 생성하여 하나의 토큰으로 묶는다.
3) 트랜지션 t의 출력 플레이스에 해당 토큰들을 도착시킨다.

다음의 예제를 통하여 본 연구에서의 페트리 네트 실행을 설명하기로 한다. <그림 2>의 페트리 네트 모델에서 각 플레이스의 용량은 1이다. 현재 플레이스 P1과 P2에는 각각 클래스 A와 B의 인스턴스인 토큰 a와 b가 있고, 두 토큰은 활동상태이다. 따라서, 트랜지션 t1이 활성화 조건을 만족하므로 즉시 발사되어 입력 플레이스에 있는 두 토큰은 t1으로 이동된다. 출력 플레이스 P3로의 아크에는 {A}가



<그림 2> 페트리 네트 실행

정의되어 있어 이동된 토큰 중 클래스 A의 인스턴스인 a가 복사되어 P3로 이동된다. 출력 플레이스 P4로의 아크에는 {B}가 정의되어 클래스 B의 인스턴스인 b가 복사 이동된다. 플레이스 P5로의 아크에는 {A,C}가 정의되어 있다. 따라서, 토큰 a가 복사되고, C클래스의 인스턴스인 새로운 토큰 c가 생성되어 이들 두 토큰이 하나로 합쳐져 P3로 이동된다. 이 새로이 합쳐진 토큰은 플레이스에서 하나의 토큰으로 간주된다. 결국 t1의 발사결과는 그림 2-(b)와 같다. 그림 2-(b)에서 t2와 t3가 활성화 조건을 만족하나 이 중 t2가 우선적으로 발사된다고 하자. 트랜지션 t2의 발사 결과 토큰 a가 복사되어 P1으로 이동하고, P4에 있던 토큰 b는 없어진다. 따라서, 그 결과 그림 2-(c)와 같다. 그림 2-(c)에서 t3가 발사되면 그림 2-(d)와 같이 되어 P5에 있던 토큰 (a, c)는 없어지고, 새로운 토큰 b가 P2에 생성된다.

3. 페트리네트 모델 간의 메시지 교환

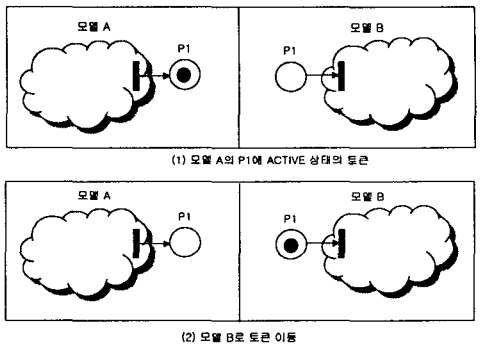
HLA/RTI 하에서 모델 간의 교신을 위한 메시지 형태는 교신 클래스(Interaction Class)와 객체(Object)로 나뉜다. 교신 클래스는 메시지가 교신을 위한 일시적인 수명을 갖는 반면, 객체는 메시지의 내용에 관계없이 계속적으로 유효하다. 예를 들어, 탱크가 페더레이트 A에서 페더레이트 B로 이동할 때 탱크는 계속 유효하여 객체에 해당한다. 그러나, 미사일을 A에서 B로 쏘았을 경우 미사일은 A에서 생성되어 B로 보내지고, B에서 미사일을 받아 처리 후 즉시 없어져 교신 클래스에 해당한다. 객체는 특성치(attribute)를 포함하고, 교신 클래스는 파라미터(parameter)를 포함한다. 이들은 FOM(Federate Object Model)에 포함되어 FED(Federation Execution Data) 파일에 정의되어야 한다.

본 연구에서 사용하는 페트리네트 이산사건 시스템에서 모델 간에 송수신되는 엔터티

는 토큰으로 물리적인 엔터티와 정보형 엔터티로 나뉠 수 있다. 정보형 엔터티는 생산시스템의 경우 콘트롤러와 기기간에 주고 받는 명령 또는 상태정보가 그 예로써 교신 클래스의 정의에 부합된다. A모델에서 생성된 정보형 엔터티는 즉시 B모델로 이동되어 처리 후 제거된다. 물리적인 엔터티는 생산시스템에서 작업물, 공구, 차량등과 같이 물리적인 실체를 갖는 것으로 객체에 해당하나 교신 클래스로 정의하여 다음에 설명될 토큰의 복사 및 제거 절차에 따르도록 하였다. 모델간의 엔터티를 송수신하기 위하여 페트리 네트의 일반적인 플레이스외에 정보형 엔터티 입, 출력 플레이스와 물리적 엔터티 입, 출력 플레이스를 추가하였고, FED파일에 정의된 송수신 메시지의 구조는 다음과 같다.

```
(class Token reliable timestamp ToModel
  (parameter TokenID)
  (parameter TokenClass)
  (parameter TokenPlace)
  (parameter TokenCreateTime)
  (parameter EventType))
```

모델간에 정보형 엔터티를 송수신하는 절차는 <그림 3>과 같다. 모델 A에서 생성된 정보형 엔터티가 모델 B로 이동한다고 하자. 두 모델은 정보형 엔터티를 송수신하기 위하여 P1플레이스를 공유하고 있고, 모델 A의 P1은 정보형 엔터티 출력 플레이스, 모델 B의 P1은 정보형 엔터티 입력 플레이스이다. 정보형 엔터티에 속하는 토큰이 모델 A의 P1에 도착하여 활동 상태가 되면 토큰에 포함된 정보를 참조하여 토큰 메시지를 B모델로 보내고, P1의 토큰을 삭제한다(그림 3-(a)). 정보형 엔터티를 보내는 메시지의 파라미터 EventType의 값은 토큰이 P1에 도착했음을 알리는 ARRIVAL로 한다. 토큰 메시지를 받은 모델 B는 파라미터에 정의된 TokenPlace와 동일한 이름을 갖는 정보형 엔터티 입력 플레이스에 해당 토큰을 생성시킨다(그림 3-(b)).

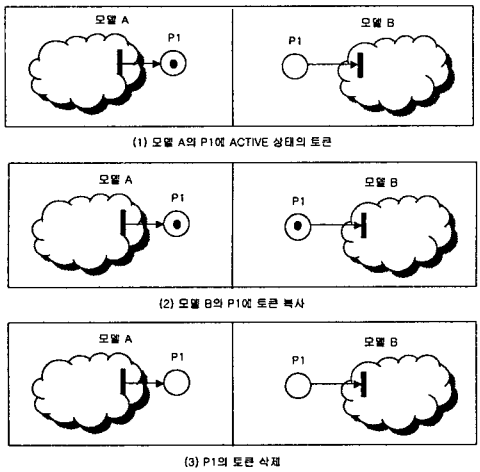


<그림 3> 정보형 엔터티 송수신

<그림 4>의 모델 간에 물리적 엔터티를 송수신하는 절차는 다음과 같다. 그림의 두 모델은 물리적 엔터티를 송수신하기 위하여 P1 플레이스를 공유하고 있다.

1) 모델 A에 토큰이 물리적 출력 플레이스인 P1에 도착하여 활동상태가 되면 토큰에 포함된 정보를 참조하여 토큰 메시지를 모델 B로 보낸다. 이 때 파라미터 EventType의 값은 ARRIVAL 로 하고, TokenPlace의 이름은 공유 플레이스의 이름인 P1으로 한다(그림 4-(a)).
2) 토큰 메시지를 받은 모델 B는 파라미터에 정의된 TokenPlace의 이름을 갖는 물리적 엔터티 입력 플레이스인 P1에 해당 토큰을 생성한다(그림 40-b)).
3) 모델 B의 P1플레이스에 있는 토큰이 트랜지션 발사에 따라 이동되면 해당 토큰 정보를 참조하여 토큰 메세지를 모델 A로 보낸다. 이 때 파라미터 EventType의 값은 DEPART로 하고, TokenPlace의 값은 P1으로 한다(그림 4-(c)).
4) 토큰 메시지를 받은 모델 A는 파라미터 TokenPlace의 이름을 가진 물리적 엔터티

출력 플레이스에 있는 해당 토큰을 삭제한다(그림 4-(d)).



<그림 4> 물리적 엔터티 송수신

RTI는 수신한 메시지의 클래스를 참조하여 이들을 수신 신청한 모든 페더레이트에게 메시지를 보내기 때문에 각 페더레이트에서 보내는 특정 메시지의 수신자를 정의할 필요가 있다. 이는 RTI의 데이터 분배 서비스를 이용하여 송수신할 메시지의 전달범위(region)을 정의할 수 있다. 예를 들어, 3개의 페더레이트가 있고, 3 종류의 메시지를 표 1과 같이 송수신한다고 하자. 3개의 전달 범위를 생성하여 다음과 같이 서로 중첩되지 않는 범위를 할당하고, 각 페더레이트의 수신범위를 [표 1]의 마지막 열에 나타난 바와 같이 설정한다.

Region1의 범위: 0 이상 1 미만
Region2의 범위: 1 이상 2 미만
Region3의 범위: 2 이상 3 미만

각 페더레이트는 <표 1>의 수신범위에 해당하는 메시지 만을 수신하기 위하여 subscribeInteractionClassWithRegion 서비스를 요청한다. 또한, 각 메시지의 송신시 sendInteraction WithRegion 서비스를 이용하여 메시지의 송신 범위를 설정한다. 예를 들어, <표 1>에서 MSG12의 수신자는 페더레이트 2 이므로 송신범위를 Region2로 한다. 구체적으로 InteractionClass에 정의된 Token 클래스의 전달범위인 ToModel의 값을 Region2로 한다.

<표 1> 페더레이트 간의 메시지 송수신

페더레이트	송신메세지	수신메세지	수신범위
Federate1	MSG12	MSG31	Region1
Federate2	MSG13	MSG12	Region2
Federate3	MSG31	MSG23	Region3

4. 시뮬레이션 시각 동기화

시뮬레이션 시각 동기화 방법을 위해 RTI의 시간관리 서비스는 크게 두가지로 TSO(Time stamped order)메세지 전달 서비스와 페더레이트가 자신의 시뮬레이션 시각(local time)의 진전을 요청하고, 요청된 시각 이전의 외부사건이 더 이상 없을 때 시각 진전을 허락하는 프로토콜을 제공한다. 이 두가지 일을 위한 절차는 다음과 같다.

1) 페더레이트는 RTI에 TSO 메시지를 보낸다.
2) RTI에 도착한 TSO 메시지는 큐에 시간순으로 저장된다.
3) 페더레이트가 다음 외부 사건을 요청하면 RTI는 큐의 가장 빠른 시각을 갖는 메시지가 더 이상 없거나 다른 페더레이트로부터의 지연된 이른 시각의 메시지가 없을 때 TSO 메시지를 해당 페더레이트로 보낸다. 다른 페더레이트로 부터의 지연된 이른 시각의 메시지 여부에 대한 결정은 각 페더레이트로부터 보내어질 메시지의 최소한의 시각인 LBTS(Low nound time stamp)에 따른다.
4) RTI는 페더레이트에 시뮬레이션 시각 진전을 허락하는 Time Advance Grant(TAG) 서비스를 보낸다.

다음 사건 시뮬레이션(next event simulation) 엔진을 갖는 페더레이트가 시각 T에서의 내부 사건을 처리 후 외부사건을 요청하는 서비스는 NER(Next Event Request) 또는 NERA(Next Event Request Available)에 의한다. 이 두 서비스는 보수적 방법에 속하여 다음 사건이 발생될 최소한의 시간 간격인 lookahead 값을 필요로 한다. 그러나, lookahead 값을 0으로 하여 NERA 서비스를 요청하면 시각 T에서의 TAG가 주어졌을 경우 T와 동일한 시각의 TSO를 모두 다 받았다는 것을 보장하지 않지만 시각 T에서 메시지를 보낼 수 있다. 역으로 lookahead를 0으로 한 NER의 경우 T에서의 TAG는 동일한 시각에서의 메시지를 모두 받았다는 것을 보장하지만 같은 시각에서의 메시지 송신은 불가능하다. 따라서, NER(T)는 외부로의 메시지 송신이 항상 T이후에 발생한다는 상황에서 사용할 수 있다. 만약, 한 페더레이트에서 다음 내부사건 시각이 T이고, 이 사건의 발생은 동일한 시각에서의 외부 사건을 발생시킨다면 NERA(T)를 사용하여야 한다. 그러나, 동일한 시각에서 메시지를 수신한 페더레이트가 메시지를 그 시각에서 다시 보낸다면 이는 문제를 발생시킨다. NERA에서는 시각 T에서의 외부 메시지를 모두 받을 수 있다는 것을 보장하지 못하기 때문이다.

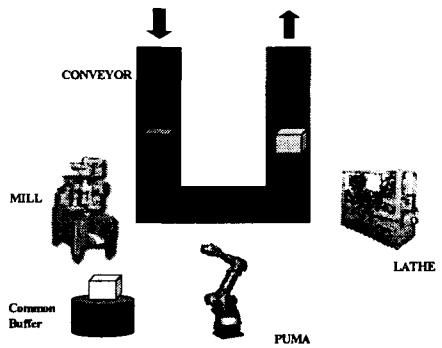
본 연구에서 이용된 이를 해결하는 방법은 다음과 같다. 페더레이트가 외부로 부터의 메시지를 받았을 때 동일한 시각에서 즉각적인 외부로의 메시지를 보내지 않도록 하는 것이다. 정보형 엔터티의 경우 수신된 메시지에 의해 즉각적인 출력 메시지를 보내는 경우를 배제한다. 이는 수신된 메시지의 처리 시간과 메시지 전송 시간을 감안하여 입력 메시지가 어느 정도 지연시간 후에 출력 메시지를 발생시키도록 한다. 물리적 엔터티의 경우 수신된 엔터티에 의한 즉각적인 사건 유발이 즉각적인 출력 메시지를 발생시킬 수 있다. 예를 들어, A모델과 B모델은 작업물의 저장을 위한 버퍼를 공유하고 있다고 하자. 작업장으로의 유입을 나타내는 A모델에서의 작업물이 버퍼에 들어오면 즉시 로봇을 나타내는 B모델로 작업물 유입 메시지를 보낸다. B모델은 이 메시지를 이용하여 작업물을 버퍼에 생성하고, 즉시 작업물 유입 메시지를 콘트롤러를 나타내는 C모델로 보낸다. 콘트롤러는 작업물을 기계로 이동시키기 위한 명령을 B모델로 보내고, B모델은 이 명령을 받아 작업물을 이동시킨다. 동시에 버퍼에서의 작업물 제거 메시지를 A모델로 보낸다. 이 모든 것이 동시에 일어난다면 NERA를 이용할 수 없다. 그러나, 실제적으로는 이 모든 것이 동시에 이루어 지지는 않는다. 버퍼에 들어온 작업물이 즉각적으로 기계로의 이동이 허용된다해도 이동되기 까지의 지연시간이 있기 마련이다. 즉, 물리적인 엔터티의 모델 간 인터페이스 공간에 약간의 지연시간을 부가할 수 있다.

본 연구에서는 이러한 방법을 이용하여 외부에서 입력되는 메시지에 의해 페트리 네트 모델의 플레이스에 토큰이 생성되면 지연시간을 갖도록 하였다. 또한, 미래사건 리스트는 외부로의 메시지 송신 사건을 포함하기 때문에 lookahead를 0으로 하고, NERA 서비스를 이용하였다. 페더레이트에서 시뮬레이션을 위한 메인 루틴은 다음 절차에 따른다.

1) 현재 시뮬레이션 시각 T에서의 모든 사건을 발생시킨다. 외부로의 사건이 발생하면 이를 출력 큐에 넣는다.
2) 출력 큐의 외부 사건들을 SendInteraction ClassWithRegion 서비스를 이용하여 RTI에 보낸다.
3) 다음 미래사건 시각인 T'을 찾아 NERA (T')서비스를 요청한다.
4) 수신된 TSO 메시지를 사건리스트에 넣는다.
5) TimeAdvanceGrant(T")을 받아 현재 시뮬레이션 시각을 T"으로 하고 1)로 간다.

5. 예제: 생산 셀 시뮬레이션

분산된 페트리 네트 모델들의 RTI 에 의한 실행을 예제를 통하여 설명하기로 한다. 자동화 생산 시스템의 중요한 요소인 셀은 <그림 5>와 같이 두 대의 기계와 로봇, 그리고 작업물의 입출력 콘베이어로 구성되어 있다. 기계는 밀링(MILL)과 선반(LATHE)으로 구성되어 있고, PUMA 로봇은 셀 내의 작업물 이동을 담당한다. 셀 내에는 하나의 작업물을 임시 저장할 수 있는 공통 버퍼가 있어 작업물의 원활한 이동을 위해 이용된다. 셀 내에 유입되는 작업물 종류는 3가지로 <표 2>와 같다.

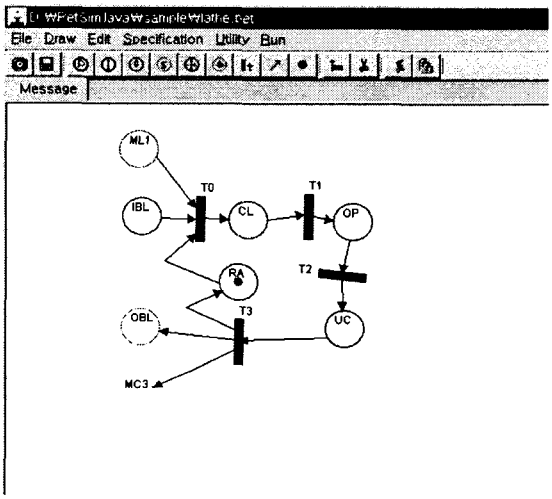


<그림 5> 유연생산 셀

<표 2> 작업물 종류

작업물	라우팅	가공시간(분)	유입비율
P1	LATHE, MILL	30, 20	1/3
P2	MILL, LATHE	20, 30	1/3
P3	MILL	20	1/3

시뮬레이션 모델은 두 기계와 로봇, 콘베이어, 그리고, 셀 콘트롤러로 구성된다. 셀 콘트롤러는 각 기기에게 적절한 명령을 하달하고, 기기의 상태를 모니터링한다. 콘트롤러와 기기 사이에 전달되는 이러한 메시지는 정보형 엔터티로 모델링 되었다. 또한, 기기 사이에는 물리적 엔터티인 작업물이 이동된다. <표 3>과 <표 4>는 각각 정보형 엔터티와 물리적 엔터티를 나타낸다. <표 4>에서 콘베이어에 각 형태의 작업물이 유입되면 이 정보(IP1, IP2, IP3)는 콘트롤러에게 전달된다. 이 정보는 정보형 엔터티의 특성을 가지나 콘트롤러 모델에서 작업물 토큰을 생성시키는 작용을 하여 물리적 엔터티로 모델링하였다.



<그림 6> 기계 모델 (LATHE)

<그림 6>부터 <그림 9>까지는 각각 LATHE, PUMA, COMVEYOR, CONTROLLER의 페트리 네트 모델을 나타낸다. 모델에서 트랜지션의 출력 아크에 정의된 토큰흐름에 대한 정의는 생략되었다. LATHE와 MILL은 동일한 운영순서를 가지고 있어 작업물이 PUMA에 의해 기계로 유입되고(즉, 작업물 토큰이 IB1플레이스에 있고), CONTROLLER로부터 작업지시 명령을 받으면(즉, 정보형 토큰이 ML1 플레이스에 있으면), 우선 작업물을 고정시키는 클램핑을 하고, NC 프로그램을 실행시켜 가공이 끝나면 작업물을 해체하는 언클램핑을 한다. 언클램핑이 끝나면 셀 콘트롤러에게 작업종료 메세지를 보낸다. <그림 6>의 LATHE 모델은 이 같은 절차를 나타내고 있고, 각 플

<표 3> 정보형 엔터티

ID	송신	수신	설명
MP1	Controller	PUMA	Mill에서 Output Buffer로 작업물 이동
MP2	Controller	PUMA	Mill에서 Lathe로 작업물 이동
MP3	Controller	PUMA	Lathe에서 Output Buffer로 작업물 이동
MP4	Controller	PUMA	Lathe에서 Mill로 작업물 이동
MP5	Controller	PUMA	Input buffer에서 Lathe로 작업물 이동
MP6	Controller	PUMA	Input buffer에서 Mill로 작업물 이동
MP7	Controller	PUMA	Common buffer에서 Lathe로 작업물 이동
MP8	Controller	PUMA	Common buffer에서 Mill로 작업물 이동
MP9	Controller	PUMA	Mill에서 Common buffer로 작업물 이동
MP10	Controller	PUMA	Lathe에서 Common buffer로 작업물 이동
MM1	Controller	MILL	Milling 작업 시작
ML1	Controller	LATHE	Lathe 작업 시작
MC1	PUMA	Controller	작업물 이동 종료
MC2	MILL	Controller	Milling 작업 종료
MC3	LATHE	Controller	Lathe 작업 종료
MB2	CONVEYOR	Controller	작업물 출발

레이스에 대한 설명은 <표 5>와 같다. 초기에 LATHE 클래스에 속하는 인스턴스 토큰이 플레이스 RA1플레이스에 있어 기계가 유휴 상태임을 의미한다.

<그림 7>의 PUMA 모델에서 초기에 PUMA 클래스에 속하는 인스턴스 토큰이 유휴상태를 나타내는 RA1 플레이스에 있다. PUMA 로봇은 콘트롤러로부터 <표 3>에 설명된 것과 같은 10가지 명령(MP1 부터 MP10)을 하달받는다. 각 명령에 따라 PUMA 토큰은 작업물을 해당 목적지로 이동시키고, 다시 RA1 플레이스로 돌아온다. 작업을 끝내면, 로봇은 CONTROLLER에게 작업 종료 메시지인 MC1을 송신한다. 작업물은 CONVEYOR, LATHE, MILL로부터 유입 또는 방출되고, 이들 모델과의 작업물 인터페이스는 <표 4>에 설명된 바와 같다.

<그림 8>의 CONVEYOR 모델은 작업물의 생성과 제거에 관련되어 있다. 초기에 각 작업물 형태에 속하는 작업물들이 P1, P2, P3

<표 4> 물리적 엔터티

ID	송신	수신	설명
IP1	CONVEYOR	Controller	작업물 P1 도착
IP2	CONVEYOR	Controller	작업물 P2 도착
IP3	CONVEYOR	Controller	작업물 P3 도착
IB	CONVEYOR	PUMA	셀의 작업물 입력 지점
OB	PUMA	CONVEYOR	셀의 작업물 출력 지점
IBM	PUMA	MILL	MILL 작업물 입력 지점
OBM	MILL	PUMA	MILL 작업물 출력 지점
IBL	PUMA	LATHE	LATHE 작업물 입력 지점
OBL	LATHE	PUMA	LATHE 작업물 출력 지점

에 있고, 이들이 셀에 유입되면 셀의 입력 지점에 해당하는 IB 플레이스로 이동된다. 동시에 CONTROLLER에게 작업물 유입 신호를 보낸다. 하나의 작업물이 IB 플레이스에 도착하면 같은 형태의 작업물이 복사되어 플레이스 P0, P2, P4로 이동되어 반복적인 작업물 생성을 가능토록 한다. IB 플레이스는 물리적 엔터티 송신 플레이스로 작업물 토큰이 도착하면 토큰 정보를 PUMA 모델에게 보낸다. 또한, PUMA로부터 작업물이 셀의 출력지점에 있다는 토큰 정보를 받으면 해당 작업물 토큰이 물리적 엔터티 수신 플레이스인 OB 플레이스에 생성되고, JD 플레이스에서 제거된다.

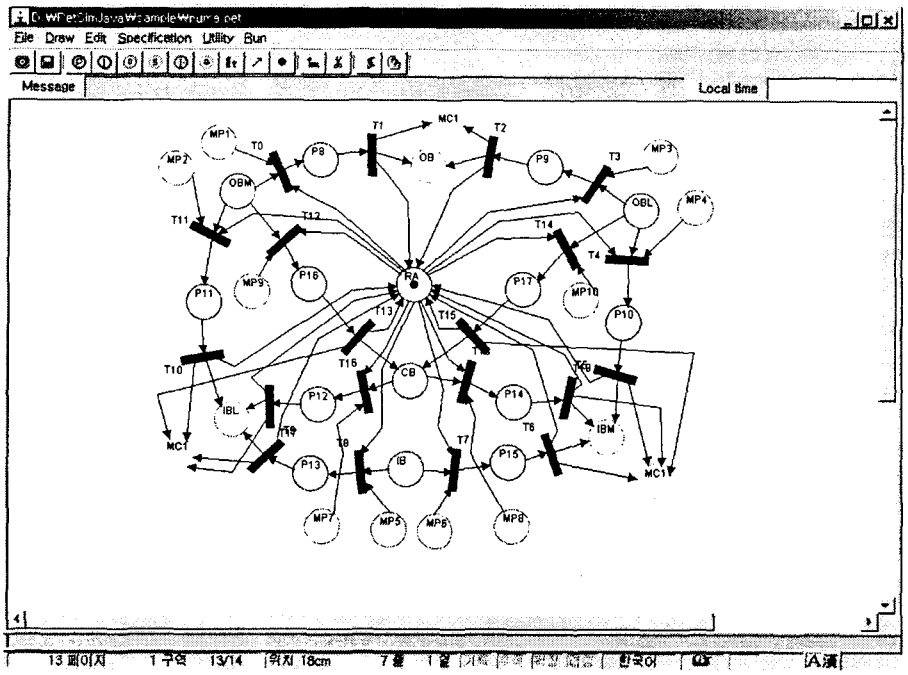
<그림 9>는 CONTROLLER 모델로서 셀 내의 자원인 LATHE, MILL, PUMA가 토큰으로 표현 되어 각각 유휴상태를 나타내는 LA, MA, PA 플레이스에 있다. 작업물이 셀에 도착하였다는 작업물 토큰 정보를 CONVEYOR로부터 받으면 해당 작업물 토큰을 IP1, IP2, 또는 IP3에 생성한다. 세가지 작업물 형태에 따른 작업순서가 페트리 모델로 나타내어 각 작업 순서에 따라 필요한 명령을 기계와 로봇에 내리도록 모델링 되었다.

위에서 설명된 각 페트리 네트 모델들은 각각의 컴퓨터에서 실행되어 컴퓨터 네트워크를 통하여 분산 시뮬레이션을 수행하였다. 부가적으로 모든 모델을 관리하는MANAGER 페더레이트가 추가되어 CRC가 실행되는 컴퓨터에 있도록 하였다. MANAGER 페더레이트는 각 페트리 네트 모델들의 RTI상에서의 페더레이트 생성, 조인, 제거, 실행 초기화 등을 위한 관리자 역할을 한다. RTI 소프트웨어는 Pitch Portable RTI (pRtiTM) [6]를 이용하였다.

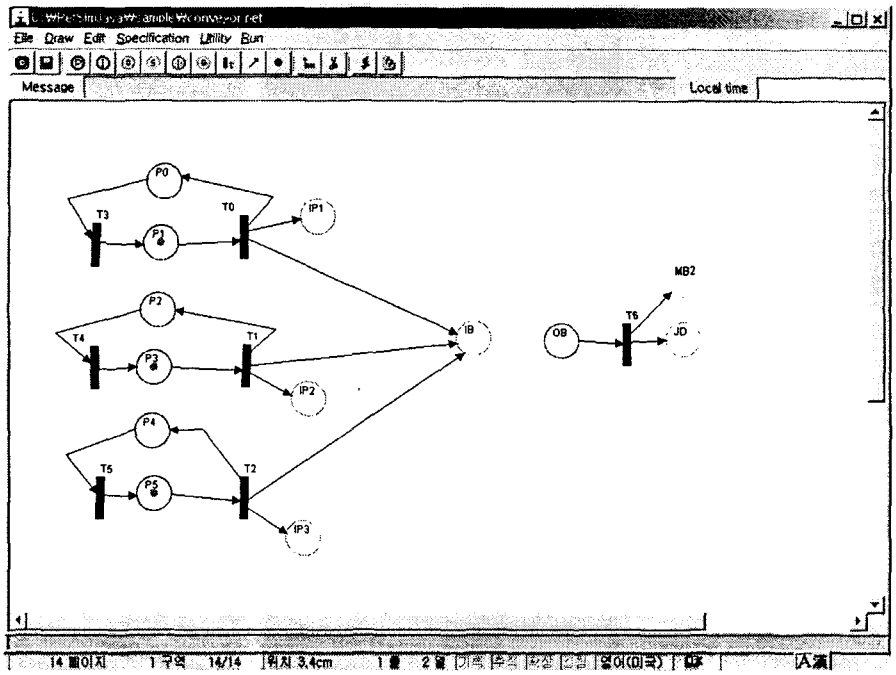
<그림 10>은 분산된 시뮬레이션 실행 결과물의 일부인 엔터티의 흐름도이다. 각 시간별로 한 모델에서 다른 모델로 보내어 지는 정보형 과 물리적 엔터티의 흐름을 나타내고, 중요 자원인 LATHE, MILL, PUMA의 작업시간을 포함한다. 예를 들어, LATHE는 시각 1.3에서CONTROLLER로 부터 작업지시 명령인 ML1을 받아 32분간 작업을 수행하고, 시각 33.3에서 CONTROLLER에게 작업 종료 메시지인 MC3를 보낸다. 이 흐름도는 모델을 검증할 수 있게 할 뿐 만 아니라 자원의 이용율 등에 대한 시스템 성능 분석을 가능케 한다.

<표 5> LATHE 모델에서의 플레이스

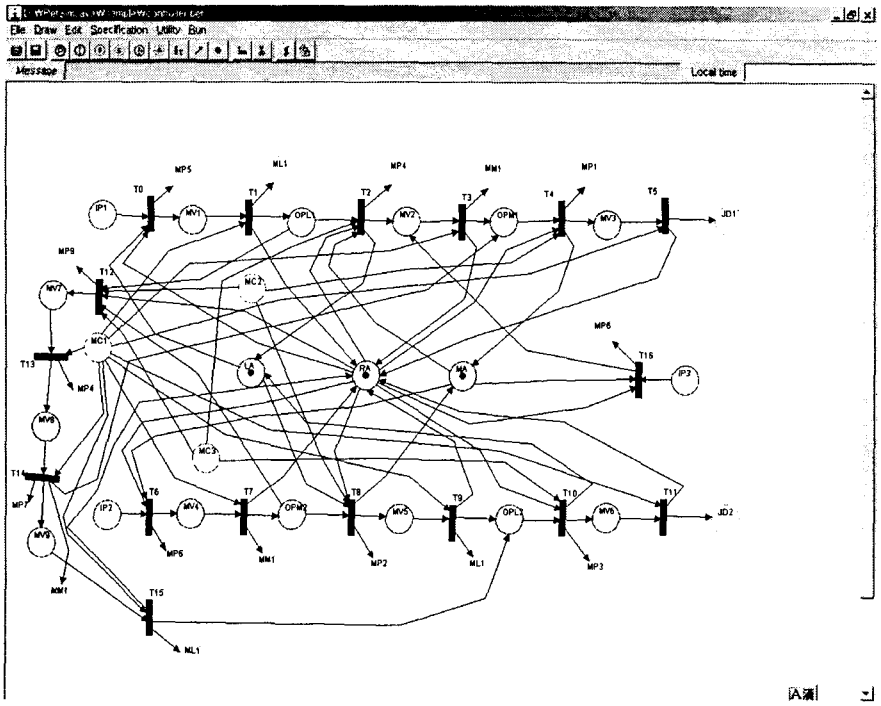
플레이스	형태	설명
ML1	정보형 엔터티 수신	CONTROLLER로부터 작업 지시 메시지수신
IBL	물리적 엔터티 수신	PUMA로부터 작업물 유입
OBL	물리적 엔터티 송신	PUMA로 작업 종료된 작업물 송출
MC3	정보형 엔터티 송신	CONTROLLER로 작업 종료 신호 송신
CL		클램핑
OP		가공 작업
UC		언클램핑
RA		기계 유휴



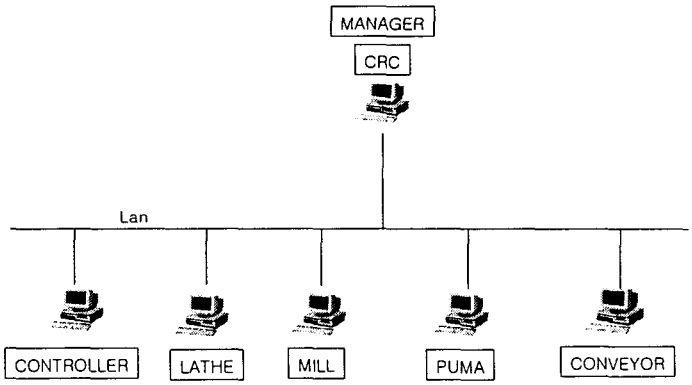
<그림 7> PUMA 모델



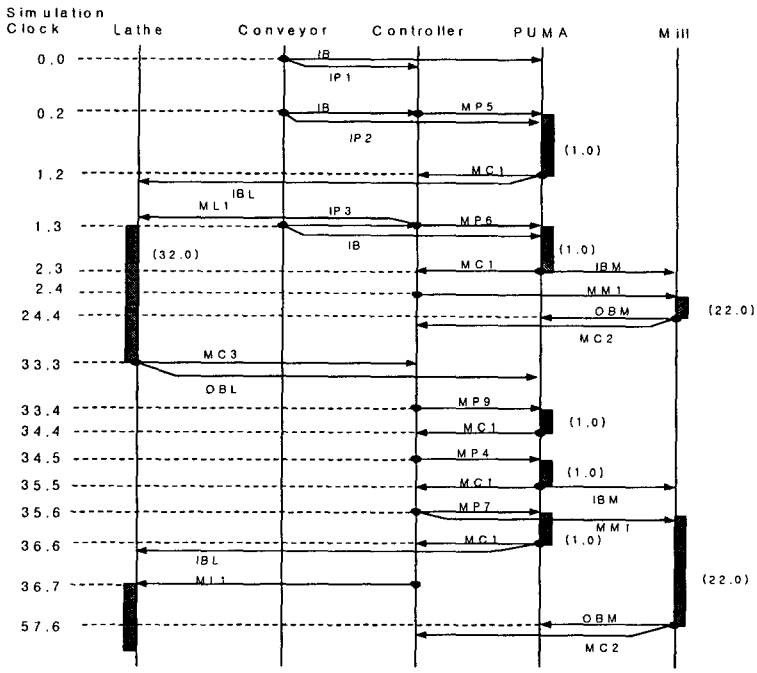
<그림 8> CONVEYOR 모델



<그림 9> CONTROLLER 모델



<그림 10> 시뮬레이션 모델의 분산



<그림 11> 모델 간 엔터티 흐름

6. 결론

분산 시뮬레이션은 실행시간의 단축뿐 만 아니라 시뮬레이션 모델의 분산이라는 측면에서도 장점을 제공한다. 특히, 복잡한 시스템을 시뮬레이션 하는 경우에 모듈화된 단위 모델을 용이하게 작성 할 수 있어 효과적인 모델링 작업을 가능케 한다. 또한, 각 단위 모듈들을 분산된 네트워크를 통해 실행시킴으로써 시뮬레이션 모델을 쉽게 검증하고, 수정할 수 있는 장점을 제공한다.

본 연구에서는 RTI를 이용하여 페트리 네트 모델의 분산 시뮬레이션을 수행한 결과를 제시하였다. 특히, 각 단위 모델간의 인터페이스 방법과 메시지 송수신 방법, 그리고, 시뮬레이션 시각의 진전을 위한 RTI 인터페이스 서비스를 어떻게 사용하는지에 대한 연구를 수행하였다.

셀 생산 시스템을 대상으로 분산 시뮬레이션을 수행한 결과 모델 간의 인터페이스는 RTI를 이용하여 매우 용이하게 구현할 수 있었고, 안정적인 실행을 가능케 하였다. 그러나, 모델 간의 시각 동기화를 위해서는 세심한 주의와 분석이 요구된다. 즉, 어떠한 객체 또는 정보들을 모델 간에 주고 받는 엔터티로 모델링할 것이지, lookahead 값을 어떻게 할당하고, 페더레이트에서 외부사건을 요청하는 서비스를 어떤 것으로 할지, 그리고, 외부로 부터의 사건에 반응하는 내부 사건이 즉각적인 외부로의 사건을 발생시키는 지를 검토하고, 결정하여야

한다.

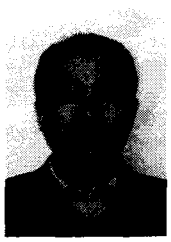
본 연구에서는 페트리 네트 모델 만을 대상으로 하였으나, 앞으로 다양한 모델링 방법에 따른 여러 종류의 시뮬레이션 모델을 분산 실행시킬 수 있는 방법에 대한 연구를 계속할 예정이다.

주 작 성 자 : 임 동 순
논문투고일 : 2004. 10. 18
논문심사일 : 2004. 11. 04(1차), 2004. 11. 05(2차), 2005. 01. 27(3차)
심사판정일 : 2005. 01. 27

참고문헌

[1] Chandy, K. M. and Misra J., "Asynchronous Distributed Simulation via a Sequence of Parallel Computations", *Communications of the ACM*, Vol. 24, No. 4 (1981), pp. 198-205.
[2] Department of Defense, High Leval Architecture Run-Time Infrastructure Programmer's Guide (Version 1.3), 1998.
[3] Fujimoto, R. M., Parallel and Distributed Simulation Systems, Wiley, 1999.
[4] Jefferson, D., "Virtual Time", *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3 (1985), pp. 404-425.
[5] Object Management Group, Inc., Distributed Simulation Systems Specification, 2000.
[6] Pitch AB, pRTI, http://www.pitch.se.

● 저자소개 ●



임동순
1983 한양대학교 공과대학 산업공학과 학사
1986 한국과학기술원 산업공학과 석사
1991 아이오와 주립대학교 산업공학과 박사
1986 ~ 1988 시스템공학연구소 연구원
1992 ~ 현재 한남대학교 산업시스템공학과 교수
관심분야 : 이산사건 시뮬레이션, 생산시스템 분석