IE Interfaces
Vol. 13, No. 3, pp. 528-538, September 2000.

생산시스템 시뮬레이션을 위한 High Level Architecture / Run-Time Infrastructure의 적용†

홍윤기[1] · 권순종[2]

[1]한성대학교 산업시스템공학부 / [2]RENU Soft (주)

An Application of HLA/RTI to Manufacturing Simulations

Yoon-Gee Hong[1] · Soon-Jong Kwon[2]

HLA is a general-purpose software architecture for distributed simulation designed to support a wide range of simulation approaches and application. The US DoD's HLA for modeling and simulation can certainly be regarded as the state of the art in distributed simulation. It is a mandatory standard for military simulation.

The purpose of this paper is to describe applications of HLA/RTI in multiple domains across the manufacturing systems society. In many and large scale industrial systems, enormous data is generated, and is to be managed in an effective way. It needs a high performance common network library. Furthermore, it must satisfy the real function of system facilities as much as possible. The RTI is an implementation of the interface specification, provided as a set of services.

Some applications focusing on the area of a small manufacturing system were demonstrated. The integration could be achieved using the HLA, together with interface modules for each of the subsystems. We have found that HLA/RTI are cable of meeting the functional requirements for a given system environment.

1. 서 론

컴퓨터 분야에서는 최근 수년 전부터 네트워크와 관련된 기술이 급속도로 발전되어 왔다. 특히 분산 시뮬레이션 분야는 이와 같은 컴퓨터 네트워크 기술을 발판으로 다양한 내용에 이르기까지 개발과 응용이 뒤따르고 있다. 분산된 시스템간의 데이터 전송을 원만하고 효율적으로 처리하기 위한 많은 연구와 방법이 소개되었다(Fujimoto, 1990). 이들 연구결과 가운데에는 시간의 동기화와 이에 따른 시뮬레이션 모듈간의 데이터 전송에 관한 문제들을 대표적으로 꼽을 수 있다(Bagrodia, Chandy, and Misra, 1987).

미국 국방성(Department of Defense; DoD)은 최근 새로운 개념의 분산 시뮬레이션 방법을 제안하고 있다(DMSO). DoD에서 제안한 분산 시뮬레이션의 신 설계구조 개념은 HLA(High Level Architecture)라 불리는 것으로, 이는 시뮬레이션 모델간의 상호운용성(interoperability), 재사용성, 그리고 확장성 등을 높이기 위한 것으로 요약된다.

지금까지는 여러 컴퓨터에 나뉘어 실행되는 시뮬레이션 모듈은 대체로 동일한 시뮬레이션 언어로 구현된 모델로 운영체제 또한 동일하거나 유사한 운영체제를 사용하여 왔다. HLA는 분산된 시뮬레이션 모델이나 모듈들 간의 상호 운용성을 보장하기 위해 제안되었으며, 특히 wargame이나 훈련 교육 등의 시뮬레이션 모델과 인간이 참여하는 시뮬레이션(human-in- the-loop)들 간의 상호작용이 원만하도록 뒷받침하는 개념을 포함하고 있다. 더욱이, 과거 분산 시뮬레이션에서 단점으로 지목되었던 이 기종 시뮬레이터 간 연동의 불가능에 대한 제약조건을 극복하였고, 시뮬레이션 개발 언어가 다르더라도 상호 호환이 가능하도록 구성되었다.

미국 국방성 산하의 DMSO(Defense Modeling and Simulation Office)를 비롯한 관련 학계에서는 HLA의 확장성 및 성능의 효과를 인지하여 최근 몇 년간 많은 노력과 연구를 경주해 오고

† 본 연구는 1999학년도 한성대학교 교내연구비 지원과제임.

있다. 이미 국방분야의 시뮬레이션에서 HLA는 필수적인 항목으로 고려되기 시작하였고, 현재는 물류의 수송이나 의료같은 민간분야의 시뮬레이션 등에도 HLA를 적용하려는 노력을 시도하고 있다(Thomas, Steffen, Ulrich, 1999). 이들의 HLA에 대한 공통적인 결론은 분산 시뮬레이션에서 HLA는 네트워크의 표준으로 사용될 것이며, 관련 학계나 산업체 전반에 미치는 영향이 지대할 것으로 예측하였다.

본 연구에서는 위에서 언급한 HLA의 개념 및 방법론을 소규모 생산시스템에 적용하여 활용 가능성과 아울러 기타 생산 관련 시스템으로의 확장성을 검토해 보고자 한다.

2. 분산 시뮬레이션 기술

2.1 분산 시스템

분산 시스템이란 사용자 인터페이스(user interface)를 사용하여, 최종 사용자(end user)에게 네트워크 통신을 통하여 서비스를 제공하기 위해 서버를 사용하면서 상호 협동하는 구성인자들의 단일 집합을 말한다(John *et al.*, 1997). 분산 시스템에서는 주로 네트워크를 이용하여 주로 지역적, 기능적 및 임무에 따라서 정의되는 원격지에 있는 시스템간의 상호작용을 수행한다. 분산 시스템은 일반적으로 distributed operating system, distributed processing system, distributed application, distributed database, 그리고 distributed file system 등과 같이 주로 5가지 형태로 구성된다(Sochats and Williams, 1992).

2.2 분산 시뮬레이션

분산 시뮬레이션은 특정 시스템의 시뮬레이션을 수행함에 있어 기능별, 임무별 등으로 나누어 서로 다른 프로세서에서 진행시켜 전체적으로 수행시간을 줄일 수 있는 시뮬레이션 방법이다. 하나의 시스템에서 발생하는 부하를 몇몇 다른 프로세서에서 진행하므로 그만큼 시스템 운영 효율을 기대할 수 있을 것이다. 또한 각각의 특성에 맞게 서로 분산시켜서 차후 발생 가능한 유지 및 보수의 관점에서 시간절약과 경제성을 꾀할 수 있을 것이다.

2.3 객체지향 모델링

최근 객체지향이라는 용어를 자주 접하게 된다. 정보가 방대해지고, 개방화되고, 인터넷에서 발생되는 추세에 기존의 자료처리 방식으로는 도저히 현재의 수요를 감당해 낼 수 없을 뿐더러, 이용하는 것조차도 자주 한계에 부딪히곤 한다. 이러한 소프트웨어의 한계를 극복하기 위해서 등장한 것이 객체지향 개념이다(Marco, 1999). 객체를 사용하는 이유는 우선 사용하기가 쉽고, 확장하기가 쉬우며, 유지 및 보수하기가 용이하기 때문이다. 객체지향 모델은 모듈(module)의 개념을 갖는다. 모듈이라는 것은 과거에 라이브러리라고 표현했던 것과 유사한 것으로, 근본적으로 다른 점은 이 모듈 안에 자료뿐만 아니라 기능(function)의 특성도 가지고 있다는 점이다.

2.4 DIS와 ADS

DIS(Distributed Interaction Simulation)는 세계 환경의 시간과 공간이 응집된 종합적 표현이 상호작용과 자유로운 오퍼레이터들의 활동을 연결하기 위해 설계되었다. 이 종합적인 환경은 분산된 자율적인 시뮬레이션 응용들 사이에 서로 실시간으로 데이터를 교환하기 위해 개발되었다. 시뮬레이션 응용은 시뮬레이션, 시뮬레이터 및 기계 장비로 구성되는데 이들은 표준 컴퓨터 통신 서비스를 통해 서로 연결되어 있다. 컴퓨터 시뮬레이션 요소들은 기능적 또는 임무별로 서로 분리되어 단일 또는 그 이상의 지역적으로 분산된 곳에 위치될 수도 있다(DMSO[2]; Todd, 1995).

DIS의 뿌리를 캐어 이해하는 데에는 우선 DoD에서 시도하여 발전시킨 SIMNET(Simulation Network)을 빼어 놓을 수 없다. SIMNET은 ARPA(Advanced Research Projects Agency)와 육군의 STRICOM(Simulation, Training, and Instrumentation Command)에서 관리되었다. 이 프로젝트는 공통의 가상환경에서, 분산된 지역에서, 오퍼레이터가 있는 훈련 시뮬레이터를 연결하여 운영·관리되었다. 이때, 오퍼레이터들은 거의 실시간(near-real-time)으로 이 공통의 환경에서 서로 상호작용할 수 있었다. 수년 동안 SIMNET은 보다 더 유연성이 증대되고 현실에 대한 반영을 최대로 하도록 여러 단계를 거치면서 발전되었다. DIS의 첫 번째 임무는 다중 지역에 분포되어 있는 여러 가지 형태의 시뮬레이터들을 서로 연결하는 기반구조를 정의하는 것이었다. 이 기반구조는 각각의 목적들을 위해, 과거 시대로부터의 기술들, 여러 공급업자들이 납품한 제품들, 여러 서비스의 플랫폼들 사이에 상호작용을 위해 구축되었다.

DIS 기반구조는 종합적인 환경 내에서 인터페이스 표준, 통신 구조, 관리 구조, 기술적 포럼 및 이 기종 시뮬레이션을 변형시키기 위한 다른 요소들을 제공하고 있다. 이러한 종합적인 환경은 설계와 프로토타입, 교육과 훈련, 테스트와 평가의 사용을 위해 전반적으로 제공한다. IEEE(Institute of Electrical and Electronic Engineers) 표준 1278은 시뮬레이션들 사이의 인터페이스를 표준화하기 위해 확립되었다. ADS(Advanced Distributed Simulation)는 DIS에서 단점으로 지적된 서로 다른 적용 범위에 있는 시뮬레이션들 간의 데이터 전송과 시간의 동기화를 맞추기 위해 등장했다. ADS는 시간 응집성, 종합적 환경하에서의 상호작용을 제공하기 위해 나타난 기술이다(Todd, 1995). 이들은 지역적으로 서로 떨어져서도 가능하며, 서로 다른 상태의 시뮬레이션들 사이에서도 상호작용이 가능하다. 일반적으로 ADS의 개념이 나온 후로 DIS는 ADS의 하나의 부분집합으로 자리잡게 되었다. ADS에서는 DIS와는 달리 virtual, live,

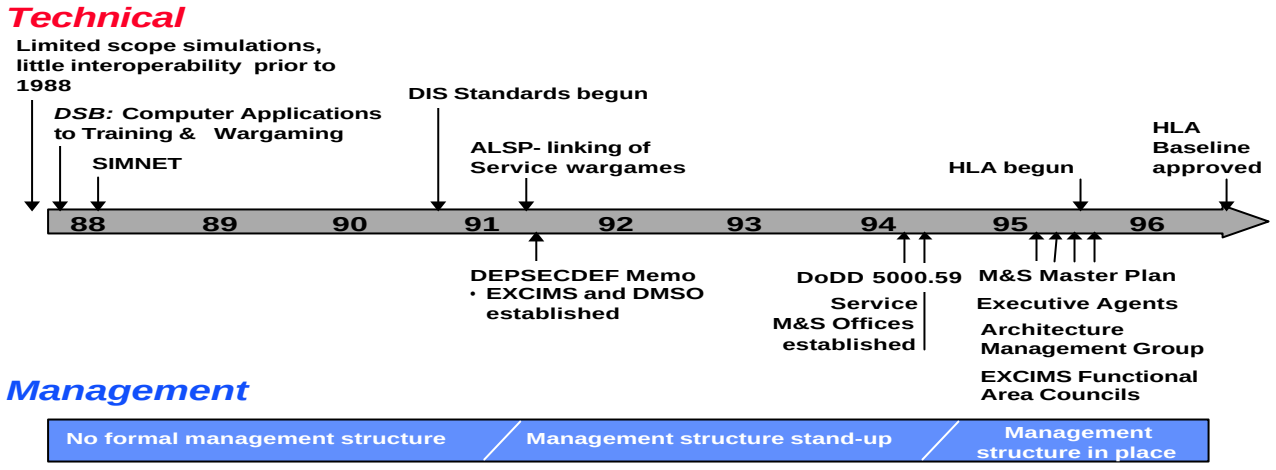

그림 1. DIS부터 HLA까지의 발전 과정.

constructive 시뮬레이션이 혼합된 시뮬레이션에서의 상호작용도 지원하게 되었다.

ADS 개념의 핵심을 이루는 것이 다음에 언급될 HLA이다. <그림 1>은 DIS부터 다음에 언급될 HLA까지의 발전 과정을 나타내고 있다. 이 그림에 따르면 1988년 SIMNET이 개발되기 이전부터 비록 소규모지만 약간의 상호작용을 지닌 시뮬레이션을 수행할 수가 있었다. 그러나 실제적인 시뮬레이션들의 상호작용은 1991년경 DIS 표준이 시작되면서부터 급속한 발전을 하게 되었다. ADS는 IEEE 1278에 기술된 표준을 넘어서는 네트워크의 구현을 포함하고 있다.

앞에서 언급한 바와 같이, DIS는 인간이 개입된 훈련(human-in-the-loop)을 그 목적으로 한다. 따라서 당연히 시뮬레이션 진행 및 관리시간은 실시간일 수밖에 없다. 이와 같은 제약으로 인해 인간이 개입된 시뮬레이션(오퍼레이터 주관하의 시뮬레이션)과 다른 시뮬레이션(컴퓨터만으로 진행되는 시뮬레이션)과는 상호작용이 용이하지 못했다. 따라서 현재는 DIS 중에서 분산이라는 개념만이 많이 사용되고 있다.

2.5 High Level Architecture (HLA)

간단히 말해서, HLA는 "시뮬레이션의 재사용과 상호작용을 위한 아키텍처"라고 말한다(DMSO[3]). HLA는 DoD에서 ADS를 구현하기 위한 하나의 아키텍처로 시작되었다. HLA의 좀더 세밀한 정의는 다음과 같다. "모든 DoD 시뮬레이션에 적합한 중요한 기능적 요소, 인터페이스 및 설계 규칙과 특정 시스템 구조에서 일반적인 기반구조를 제공하기 위해 정의된다." 앞에서도 언급되었지만, HLA는 과거 국방 시뮬레이션(SIMNET)에서, 지역적으로 떨어진 시뮬레이터들 사이에 상호작용의 취약점을 보완하기 위해 DIS가 등장하였으며, 이 DIS에 데이터 전송과 시간 동기화 등의 기능을 추가시켜 개선한 것이 ADS이다. 바로 이 ADS를 구현하는 아키텍처가 HLA이며 RTI(Run-Time Infrastructure)를 이용하여 구현하게 된다.

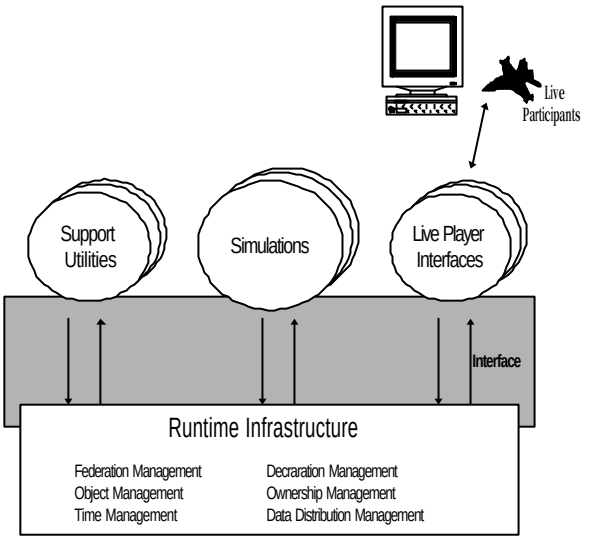

그림 2. 아키텍처의 기능적 관점.

<그림 2>는 각각의 특성을 지닌 시뮬레이션 모듈과 RTI의 관계를 아키텍처의 기능적 관점에서 바라본 그림이다. 각각의 시뮬레이션 모듈은 특정의 interface를 통하여 RTI와 상호작용하는 것이다. 일반적으로 HLA는 HLA Rule, Interface Specification 및 Object Modeling Template의 3가지 구성요소를 가지고 있다.

(1) HLA Rule

Federation이라고 불리는 시뮬레이션들의 보다 나은 상호작용을 위해 반드시 따라야 하는 규칙들이다. 이곳에 기술된 내용을 따르지 않으면, 추후 다른 federation들과의 상호작용이 어려우며 자체 federation의 확장 및 유지, 보수가 어렵게 된다.

(2) Interface Specification

RTI와 시뮬레이션 오브젝트 사이의 인터페이스 기능을 정의하고 있다.

(3) Object Modeling Template

각각의 federation과 시뮬레이션에서 다른 곳의 정보를 참조할 수 있게 사용되는 오브젝트들을 미리 선언해 두는 것이다. OMT에 기록되는 객체의 상태 중에서 중요한 것으로 Publisher와 Subscriber의 기능이 있다. Publisher는 임의의 federation 내에 선언되어 있는 객체의 상태와 속성을 다른 federation으로 전송이 가능하다는 선언이며, Subscriber는 다른 객체의 정보를 받을 수만 있고 자신이 소유한 객체의 정보를 다른 federation으로 전송할 수는 없다는 선언이다. 이 두 가지 기능이 적절하게 설정되어야 federation간의 상호작용이 정상적으로 수행될 수 있다.

특히 최근 들어 <그림2>에 나타나 있는 Live Interface의 측면이 부각되고 있다. 이미 소프트웨어 기술에서는 거의 완성단계에 접근했지만, 각종 변수를 많이 포함한 실제 세계와의 연동은 완성도가 높지 않다. 특히 실세계에서의 시간의 흐름과, 컴퓨터 소프트웨어 내에서의 시간의 흐름을 맞추는 시간 동기화(Time Synchronization)에서는 여러 외부 작용에 의해 정확한 동기화를 이룰 수 없어 부득이한 오차를 둘 수밖에 없다.

2.6 Run-Time Infrastructure(RTI)

RTI는 federate들의 실행을 도와주기 위해 필요한 소프트웨어이다. RTI는 federate들이 실행하는 동안 데이터 교환과 전체 시스템의 조정을 위해 사용된다[hla homepage-rti]. RTI는 HLA Interface Specification에 정의되어 있다. RTI는 현재 1.3v6까지 출시되었으며, DMSO는 C++, Java, CORBA, Ada 95* 등에서 사용할 수 있는 RTI 라이브러리를 배포하고 있다. 또한 RTI 운영을 위한 운영체제 역시 다양한 플랫폼하에서 사용할 수 있도록 제공하고 있다.

2.6.1 RTI에서 사용되는 용어

(1) Federation : 시뮬레이션과 공통 FOM(federation object model) 및 RTI의 집합
(2) Federate : federation의 구성요소로 하나의 시뮬레이션
(3) Federation Execution: federation의 실행
(4) Object: federation에 의해 시뮬레이션되는 영역 내의 하나 이상의 객체로, Run-Time Infrastructure에 의해 관리
(5) Interaction : 하나의 federate에 의해 발생된 시간 기록의 이밴트와 RTI를 통해 다른 것으로부터 전달받는 것
(6) Attribute : FOM에서 정의된 자료로서, 오브젝트의 클래스의 인스턴스와 관련
(7) Parameter: FOM에서 정의된 자료로서, 클래스 상호작용의 인스턴스와 관련

2.6.2 구성요소

RTI는 <표 1>에 표시된 것과 같이 크게 6가지 구성요소로 구성되어 있으며 각각의 Management에서 전체 RTI의 운영에 필요한 기능들을 수행하고 있다.

표 1. HLA Interface Specification

구성요소	기 능
Federation Management	시뮬레이션 모듈이 RTI와 연계해서 작동할 수 있는 기본 환경의 생성과 소멸 등에 대한 내용으로 구성
Declaration Management	다른 시뮬레이션 모듈들과 상호작용을 할 수 있는 기반 구조를 정의
Object Management	전체적으로 RTI와 시뮬레이션 모듈이 서로 상호 작용을 할 때 사용되는 각종 객체(object)의 생성과 에 따른 속성들을 정의
Owner Management	객체 속성의 소유권을 다른 곳으로 변경할 수 있는 기능
Time Management	각 시뮬레이션 모듈들 간의 시간적 동기화와 관련된 기능
Data Distribution Management	실제적인 시뮬레이션 모듈들 사이에서 데이터의 전송과 수신 등에 관한 기능

3. 적용 대상 생산시스템 모델

3.1 생산시스템에서 분산 시뮬레이션의 필요성

최근 생산분야 뿐만 아니라 기타 산업분야에서 시스템 운영 효율을 높이기 위해 시스템을 모듈화하여 분리 운영을 시도하고 있다. 특정 시뮬레이션 결과가 시스템 내 기타 시뮬레이션에 중요한 입력변수가 되는 경우를 흔히 접할 수 있기 때문이다. 특히 생산분야에 있어서는 생산시스템 형태상 대부분 이전 생산 모듈에서 발생한 출력 결과가 다음 생산 모듈의 중요 입력요소가 되는 것이 일반적이다. 이와 같이 다수의 생산 모듈들 사이에 원활한 상호작용이 이루어지는 것이 대단히 중요하고 이를 위한 일련의 조치가 필요하다.

생산 관련 시스템 운영에 분산시스템의 개념을 도입한 사례는 주변에서 쉽게 발견할 수 있다. 그러나, 이들 모델에서는 시스템 자체의 효율성, 경제성 측면에서의 연구 목적보다는 데이터 전송이나 시간을 관리해 주는 기능에 오히려 비중이 커지거나 관심을 갖게 되는 경우를 종종 접하게 된다.

특정 생산시스템이 분산시스템으로 정의되고 이를 운용·관리하기 위한 방법으로 시뮬레이션을 채택하여 모델을 개발하려 한다고 가정하자. 이때 네트워크 부문을 담당하는 공통의 라이브러리를 적용하여 이를 개발에 적극 활용하면 개발시간뿐만 아니라 시스템 운용 측면에서 상당한 효율을 얻을 수 있을 것이다. RTI는 이러한 공통의 네트워크 라이브러리를 제공할 수 있다. 시스템 구축을 시도하는 과정에서 이러한 기능을 지니고 있는 RTI를 객체지향적인 특성을 사용하여 손쉽게

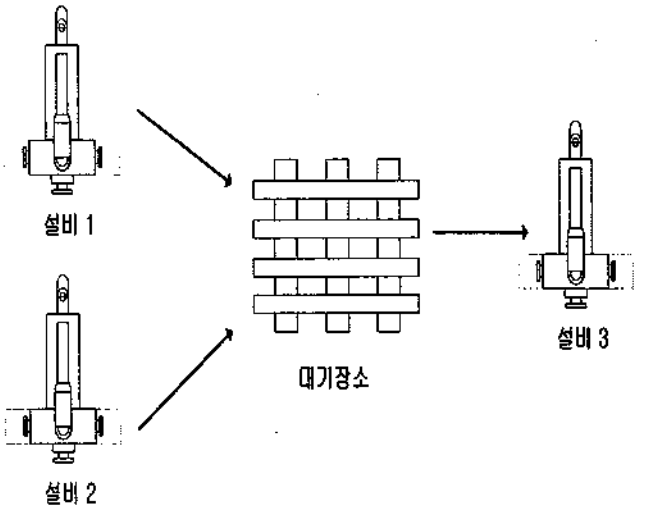

그림 3. 적용 대상 생산시스템 모델.

사용할 수 있다.

3.2 적용 대상 생산시스템의 구조

본 연구에서는 시스템의 복잡성을 최소화하는 지극히 단순한 생산시스템을 적용 대상 모델로 한다. RTI는 여러 가지 기능을 포함하고 있으나, 가장 핵심적인 기능은 바로 분산된 시스템 또는 federation 간의 원활한 상호작용이다. 따라서 본 연구에서 사용된 모델은 federation들 간의 상호작용에 그 연구중심을 맞추었다.

본 연구에서는 <그림 3>에 나타나 있는 생산시스템을 적용 대상으로 하였다. 각각의 생산 활동과 관련된 기계 및 모듈은 이론적으로 개수의 제한이 없으며, 시뮬레이션 실행 도중에도 참여 및 이탈이 가능하다. 각각의 설비는 자체적으로 일정한 생산율을 가지고 있으며, 대기장소에는 각 설비로부터 생산된 제품을 대기시킬 정해진 공간이 지정되어 있고, 만일 생산한 제품을 대기장소에서 수용할 수 없을 경우 대기장소는 선행 설비에 이 사실을 통보하여 생산을 멈추게 한다.

3.3 OMT 구축

OMT에는 시뮬레이션에서 사용되는 모든 객체의 이름과 속성을 표시하며, 속성에 관련이 되는 데이터의 종류 또한 기록을 한다. 더욱이, OMT에는 RTI에서 발생하는 각종 이벤트에 관련된 함수 및 이 함수들이 반환하거나 사용하는 데이터의 종류도 포함하게 된다.

본 연구에서 사용하는 OMT 개발 소프트웨어는 AEgis Research Corporation에서 개발하여 HLA의 공식적인 OMT 개발 툴로 되어 있는 OMDT(Object Model Development Tool)를 사용하였다. 추가적으로 Pitch Kunskapsutveckling AB사에서 개발한 Visual OMT를 덧붙여 사용하였다(Pitch Kunskapsutveckling AB; The AEgis Technologies Group, Inc.).

본 연구에서 사용하는 오브젝트는 크게 3가지의 종류로 나눌 수 있으며, 그 형태와 세부 속성은 <표 2>와 <표 3>에 기술되어 있다.

설비객체는 현재 생산이 가능한 제품의 종류에 대한 속성을 포함하고 있다. 두 번째로 사용되는 객체는 대기장소 객체이다. 대기장소에는 선행 설비객체에서 생산한 제품을 보관하고 있으며, 후위 설비에서 제품의 요청이 있을 경우 보관하고 있는 제품을 제공하는 기능을 수행한다. 특히 대기장소에는 각 제품별 보관장소 규모의 상한과 하한을 가지고 있어서 특정 수준에 도달하면 그 정보를 선행 설비로 전송하여 생산에 적용하게 한다. 그리고, 마지막으로 사용되는 객체는 시뮬레이션 모듈인 federation들 및 federation 간의 상호작용에 개입하는 객체들이다. <표 2>와 <표 3>에서 보여주는 Object Class Structure Table은 전체 시뮬레이션에서 사용하는 객체들의 구조를 나타내며, 이것을 바탕으로 하여 federation execution 내에서 객체를 생성한다.

(1) 설비 객체 속성(Facility Object Attribute)

<표 2>에는 본 연구의 대상이 되는 모델에서 설비에 대한 속성을 나타내고 있다.

표 2. Facility Object Attribute

속성 이름	데이터 형	비 고
Name	문자열	생산 객체 이름
Product	문자열	생산 가능한 제품의 종류
Result	정수	생산한 제품의 수

(2) 대기장소 개체 속성(Queue Object Attribute)

<표 3>은 생산된 제품을 보관하게 될 대기장소의 속성을 표시한다.

표 3. Queue Object Attribute

속성 이름	데이터 형	비 고
Name	문자형	대기장소 객체의 이름
P1_Count	정수	제품 1의 개수
P2_Count	정수	제품 2의 개수

OMT 개발시 가장 중요하게 염두에 두어야 할 것은 바로 publishable이라는 속성과 subscribable이라는 특징이다. 이 두 가지 특징은 object를 다른 federation에서 참조할 수 있게 하거나, 참조하게 한다.

(3) Publishable

특정의 object class는 RTI의 Publish Object Class 서비스를 이용하여 참여한 다른 federate에 속성을 전달할 수 있게 한다. 다시 말해, publish는 자신의 정보를 다른 federation에 전송할 수 있는 능력을 말한다.

(4) Subscribable

Federate는 특정 class에서 object의 정보를 잠재적으로 재사용하거나 활용할 능력이 있다. Subscribe는 다른 federation의 정보를 수신할 수 있도록 해 주는 능력을 말한다. Publish는 RegisterObjectClass라는 함수를 사용하여 RTI에 자신이 전송하게 될 데이터 및 속성의 구조를 등록하며, Subscribe는 Discover ObjectClass를 사용하여 등록된 데이터 및 속성의 구조를 알게 되고, federation Execution에 참여한 다른 federation을 참조할 수 있게 된다.

3.4 네트워크 환경 설정

각각의 시뮬레이션 모듈, 즉 federate들은 네트워크로 연결된 독립 컴퓨터에서 실행된다. Federate에서 수행된 시뮬레이션 결과는 RTI를 통해 다른 federate로 전송되며, 이를 수신한 federate는 수신된 자료를 바탕으로 또 다른 시뮬레이션을 수행한다. 이와 같이 서로 분산된 federate들 사이의 시간 동기화는 전적으로 RTI에서 관리하며 적절한 데이터의 전송 역시 RTI에서 송신 및 수신하게 된다. 이 자료들의 이동은 네트워크를 이용하여 송신 및 수신하게 되는데, 네트워크의 구성은 기본적으로 LAN(local area network)을 사용한다. 그러나 인터넷과 같은 TCP/IP를 사용하는 네트워크 환경에서도 무리 없는 진행이 가능하도록 RTI는 구성되어 있다.

4. 생산시스템 구현

4.1 개발 환경

본 모델에서 사용한 RTI 라이브러리는 DMSO에서 기본적으로 제공하고 있는 RTI Library 중에서 C++용 라이브러리를 사용하였으며, 또한 IBIS Research 사에서 개발한 IBIS RTI Adapter를 사용하였다. IBIS RTI Adapter는 C++, Ada 95, Java, CORBA로 제공되는 RTI 라이브러리를 Visual Basic 사용자와 Delphi 사용자를 위해 Active X 컨트롤 컴포넌트 형식의 라이브러리로 변형하여 제공된다.

본 연구를 위해 개발한 Application은 크게 3가지 종류로 구분할 수 있다. 기본적으로 생산에 직접적으로 참여하는 설비 federation은 마이크로소프트사의 Visual C++를 이용하여 개발하였으며, RTI의 확장성을 포함하기 위해 대기장소와 관련된 federation은 Active X를 이용하여 델파이로 개발하였다. 기본적인 시뮬레이션의 가동을 위해 개발된 두 가지 federation에 덧붙여 전체 시뮬레이션 진행상황을 감시하는 역할은 대기장소에서 수행될 수 있도록 개발되었다.

4.2 실험 환경

우선적으로 RTI의 정상적인 가동을 위해서, 하나 이상의 federation을 소유하고 있는 컴퓨터는 네트워크 상에서 분리가 되어 있어야 한다. 실험 환경에서는 각 federation은 모두 다른 컴퓨터에 설치가 되었으며, 네트워크의 사용은 LAN을 사용해도 상관없으나, 지역적인 제한이 없음을 입증하기 위해 인터넷망으로 많이 사용되고 있는 TCP/IP를 사용하였다.

4.2.1 생산설비

앞의 OMT 개발 부분에서 언급했듯이 생산설비는 각각의 생산 가능한 제품에 대한 정보를 포함하고 있다. 기본적인 속성으로는 생산에 소요되는 시간과 관련하여 생산시간 분포, 분포에 필요한 모수들, 그리고 제품 한 단위의 생산 완료를 알리는 속성 등을 가지고 있다.

4.2.2 대기장소

대기장소는 각각의 제품이 보관될 하위 대기장소를 가지고 있으며, 대기장소의 규모가 일정수준에 도달하게 되면 생산설비로 이 사실을 알려주게 된다. 또한 대기장소 다음에 위치하는 설비에서 생산이 가능하도록 제품을 전송하여 준다.

4.2.3 출력변수

출력변수의 감시는 주로 모니터를 수행하는 application에서 알게 된다. 모니터에서는 전체적인 federation execution에 참여한 각각의 federation에서 벌어지는 상황을 감시한다. 상호 인지하는 출력변수는 각 생산설비의 상태, 대기장소의 현재 규모, 그리고 생산설비에서 현재 생산하는 제품의 종류 등을 포함한다.

4.2.4 분산 시뮬레이션 개발시 RTI 도입 절차

(1) RTI 환경 초기화 및 접속

우선 기본적인 시뮬레이션을 모듈별 및 기능별로 구분하는 것이 필요하다. 이렇게 구분된 것을 일반적으로 federate라고 부른다. 각각의 federate는 다른 federate들과의 상호작용을 위해 RTI interface에 연결된다. federate와 RTI interface가 연결된 것을 federation이라고 부른다.

기본적으로 federation은 실제적인 시뮬레이션을 수행하기 전에 공통의 환경을 만들어야 한다. DMSO에서 제공하는 RTI Library에는 이들 공통의 환경 구축을 위해 rtiexec.exe 파일을 포함하고 있다. 각각의 federation을 rtiexec를 통해 기본 object의 생성과 상호작용에 요구되는 object 생성을 준비하게 된다. 일단 rtiexec가 실행된 후 <그림 4>에서 보는 바와 같이 federation은 CreateFederationExecution을 이용하여 federation에 맞는 환경을 구축한다. 이때 발생되는 공통의 환경을 fedexec라고 부른다. fedexec는 Federation Executive를 달리 부르는 말로, 여러 개의 federate를 관리한다. CreateFederationExecution에서는 미리 개발된 OMT와 이미 설정된 Federation Execution Name을 기준으로 하여 공통의 환경을 마련하게 되는 것이다.

JoinFederationExecution은 위에서 생성된 FederationExecution의 이름과 자신의 Federation Name을 이용해서 연결하게 된다.

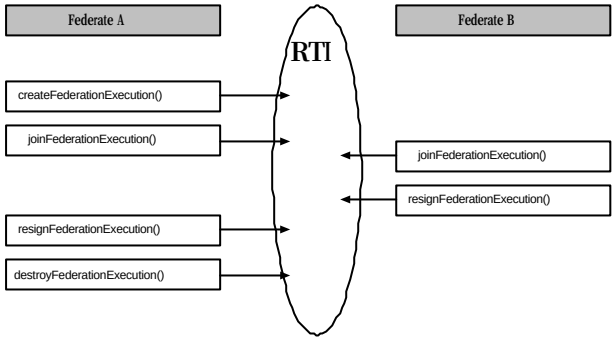

그림 4. Federation Management Life Cycle.

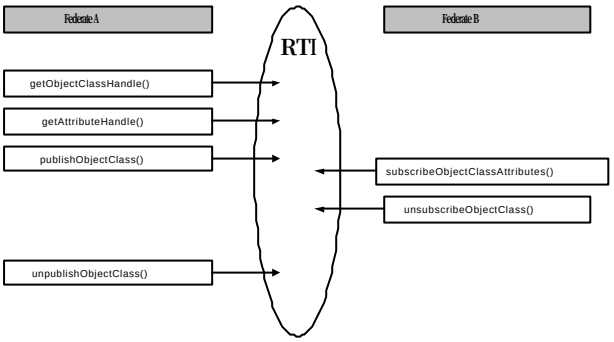

그림 5. Object Publication and Subscription.

표 4. Federation Execution의 생성과 연결, 연결해제와 소멸의 응용

```
rtiAmb.createFederationExecution( fedExecName, "Facility.fed" );
rtiAmb.joinFederationExecution(
 (char* const) myFacility->GetName(), fedExecName, &fedAmb);
rtiAmb.resignFederationExecution(
 RTI::DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES );
rtiAmb.destroyFederationExecution( fedExecName );
```

<그림 5>에서는 연결시 추후 사용하게 될 OMT에서 기술된 각 Object와 Attribute에 해당하는 ID를 부여받는 개념을 나타낸 그림이다. 이때 부여받은 ID를 기준으로 하여 다른 federation과의 상호작용을 하게 된다. 추가적으로 Interaction class에 대한 handle도 부여받는데, 이는 객체에 대한 상호작용 이외의 기타 메시지 전송 등과 같은 상호작용에 사용하게 된다. <표 4>와 <표 5>는 위에서 언급된 여러 가지 함수들이 실제로 어떠한 방식으로 사용되는지를 표시한 것이다.

(2) 데이터 전송을 위한 오브젝트의 속성 설정

<그림 5>에서는 한편 publish와 subscribe가 수행되는 과정을 표시한 것이다. publish에서는 자신이 다른 federation에게 전송할 object의 종류 및 attribute들의 크기를 가지고 PublishObjectClass를 사용하여 federation execution에 등록한다. 이렇게 등록이 완료되어야 다른 federation에서 데이터를 참조할 수 있게 된다. 이와 덧붙여 subscribe에서는 자신이 수신하게 될 object의 종류 및 attribute의 크기를 가지고 SubscribeObjectClass

표 5. Publishable과 Subscribable 속성을 위한 예제

```
ms_facilityTypeId =
            ms_rtiAmb->getObjectClassHandle(ms_facilityType6tr);
ms_nameTypeId = ms_rtiAmb->getAttributeHandle(
                              ms_nameType6tr, ms_facilityTypeId);
ms_productTypeId = ms_rtiAmb->getAttributeHandle(
                           ms_productType6tr, ms_facilityTypeId);
ms_resultTypeId = ms_rtiAmb->getAttributeHandle(
                              ms_resultType6tr, ms_facilityTypeId);

facilityAttributes = RTI::AttributeHandleSetFactory::create(3);
facilityAttributes->add( ms_nameTypeId );
facilityAttributes->add( ms_productTypeId );
facilityAttributes->add( ms_resultTypeId );
ms_rtiAmb->subscribeObjectClassAttributes(
                           ms_facilityTypeId, *facilityAttributes );
ms_rtiAmb->publishObjectClass(
                           ms_facilityTypeId, *facilityAttributes);
```

Attributes를 사용하여 마찬가지로 federation execution에 등록한다.

Publish와 Subscribe가 완료되면, 특히 Subscribe가 완료된 후 다른 federation에서 등록(publish)하게 되면 자동적으로 subscribe를 등록한 federation으로 참가 여부의 정보가 전송된다. 이때 작용하는 것이 DiscoverObjectInstance이다. 이 기능에서는 다른 federation이 등록했을 때, 그때 사용한 객체의 이름, 객체 클래스 ID 및 객체의 Instance ID를 수신한다.

이 모든 과정이 수행되어야 federation의 상호작용을 위한 기반구조의 설정이 완료되는 것이다. 이후에 각 federation에서 발생한 데이터를 공포하고 수신하는 기능이 수행된다.

지금까지의 과정이 완료되면 <그림 6>과 같은 구조가 생성된다.

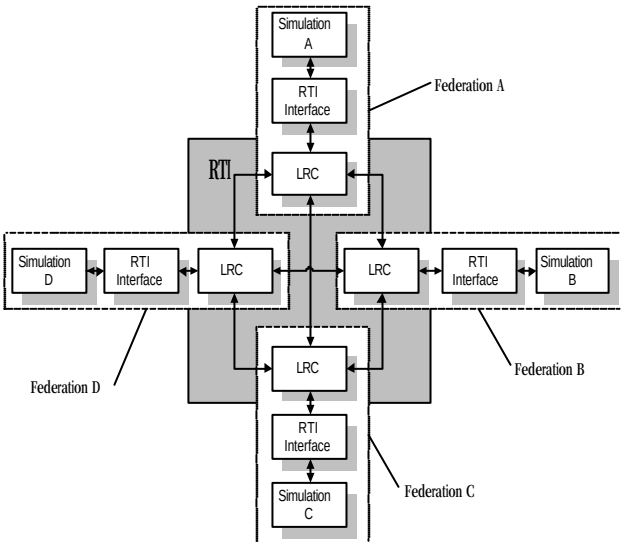

그림 6. Simulation/RTI Interface Module을 포함한 Federation의 예 (LRC : Local RTI Component).

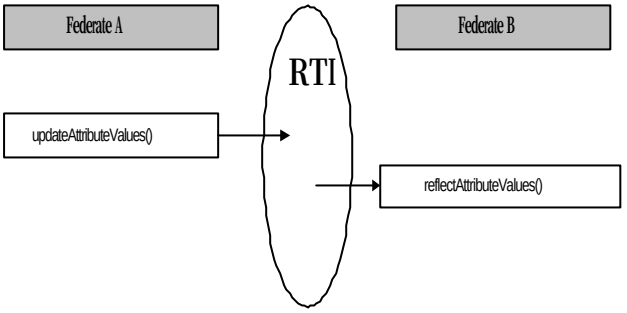

그림 7. 속성 데이터의 변경시 전송과 수신.

표 6. 데이터의 전송과 수신을 위한 예제

```
(void) rtiAmb->updateAttributeValues( this->GetInstanceId(),
    *pNvpSet, this->GetLastTimePlusLookahead(), NULL );
theAttributes.getValue( i, (char*)result, valueLength );
```

(3) 데이터 전송

시뮬레이션 수행 중 다른 federation으로 공포해야 할 메시지나 object의 attribute의 변경이 발생하면, *UpdateAttribute/alues*의 기능을 이용한다. 이 기능은 attributeHandleValuePairSet과 자신이 가지고 있는 Object Handle을 이용하여 다른 federation으로 정보를 전달하는 임무를 수행한다. 다른 federation으로 메시지나 Object의 값을 전송하기 위해서는 앞에서 언급한 publish의 기능이 수행되어 있어야 한다.

반면에 데이터의 수신 측에서는 미리 subscribe의 기능이 수행되어 있어야 하고, 다른 federation에서 데이터의 전송이 있으면 *ReflectAttribute/alues*의 기능을 수행하여 전송 받은 데이터를 분류한다. <그림 7> <표 6>은 데이터를 전송하기 위한 단계와 수신 후 원하는 정보를 추출하기 위한 과정의 예시이다.

지금까지 설명한 federation의 생성부터 소멸까지의 과정이 <그림 8>에 나타나 있다.

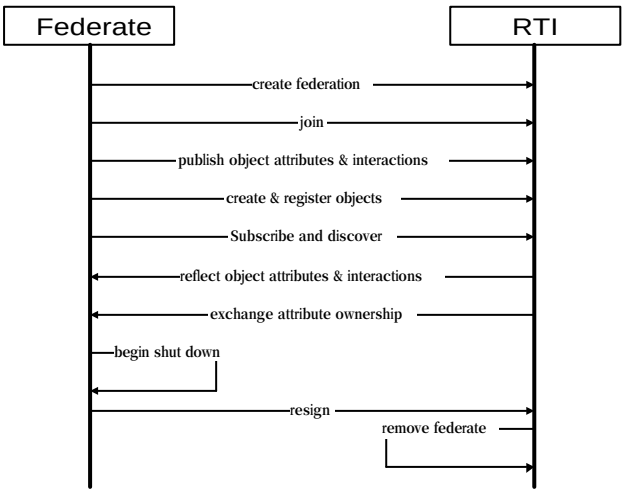

그림 8. Federate - Federation Interplay.

(4) 시간 동기화

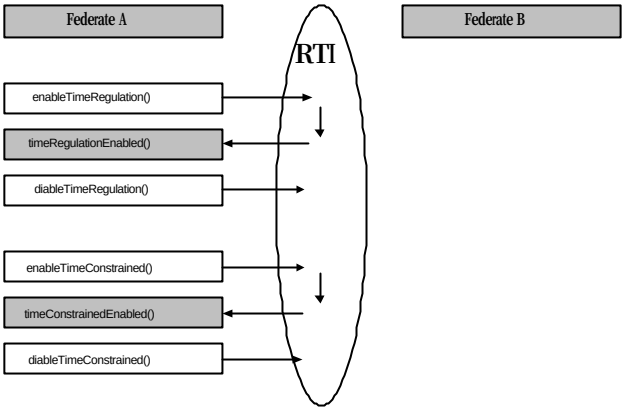

그림 9. "regulating" 과 "constrained"의 상태 변경.

정보 송수신과 더불어 RTI 및 분산 시뮬레이션에서 가장 중요한 항목이 바로 시간의 동기화(Time Synchronization)이다. 이러한 시간의 동기화 문제를 RTI에서는 Time Management 항목에서 처리한다. 그 중에서 특히 강조되는 항목으로는 regulating과 constrained이다. <그림 9>는 regulating과 constrained의 과정을 표시한 그림이다.

- Regulating : 자신을 Regulating의 성격으로 선언한 Federate는 Time-Stamp-Order(TSO) 이벤트를 발생할 수 있는 능력을 보유하고 있다. TSO 이벤트라는 것은 어느 특정 시간에 이벤트를 발생하는 것을 말한다.
- Constrained : 자신을 Constrained의 성격으로 선언한 Federate는 TSO 이벤트를 수신할 수 있다. 시뮬레이션 모델의 특성에 따라서 이 두 가지 항목이 적절히 선언되어야 원활하고 적절한 시간 관리가 될 수 있다.

<표 7>은 Time Regulating과 Time Constrained가 실제로 사용되는 예시를 표시한 것이다.

시뮬레이션에서 사용되는 시간의 진행에는 여러 가지 종류가 있다. 가장 많이 사용되는 방식은 Time-Stepped Simulation과 Event-Based Simulation 방식이다. Time-Stepped 방식은 미리 설정한 시간 단위씩 시뮬레이션 시간이 증가되는 방식이다. Time-Stepped 방식을 사용하는 federate에서는 timeAdvance-Request나 timeAdvanceRequestAvailable의 기능을 사용하여 RTI에서 시간 진행의 허가를 얻는다. timeAdvanceRequest나 혹은 timeAdvanceRequestAvailable을 전송하면 timeAdvanceGrant를 수

표 7. Time Regulating과 Time Constrained를 위한 예제

```
rtiAmb.enableTimeConstrained();
rtiAmb.enableTimeRegulation(
  grantTime,  Facility::GetLookahead());
```

표 8. Time Stepped Simulation에서의 시간진행 방법 예제

```
RTIfedTime requestTime(timeStep.getTime());
requestTime += grantTime;
rtiAmb.timeAdvanceRequest( requestTime );
```

신하여 다음 시간으로 진행한다.

<표 8>은 Time-Stepped 방식에서 사용되는 RTI의 예제이다.

반면에 Event-Based 방식으로 진행되는 federate에서는 nextEventRequest나 혹은 nextEventRequestAvailable을 사용하여 마찬가지로 timeAdvanceGrant를 수신한다. <그림 10>은 각각 Time-Stepped Federate와 Event-Based Federate에서의 시간 진행 방식에서의 차이점을 나타낸 것이다.

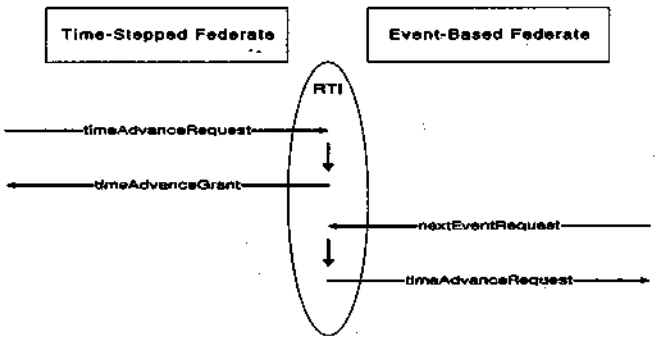

그림 10. Time-Stepped Federate와 Event-Based Federate에서의 시간진행과 관련된 함수.

4.3 시뮬레이션 수행 결과

4.3.1 설비

기본적인 생산을 담당하는 application이다. 이곳에서의 역할은 생산 가능한 제품을 event 시간에 생산을 하고 대기장소로 정보를 전달하거나 대기장소에서의 정보를 수신한다.

표 9. Federation 실행 결과

```
=> Facility Event Loop Iteration #: 26
=> Facility<0> Name: f1 Product: p1 Result: 31.28502 Time: 250
=> Facility<1> Name: f2 Product: p2 Result: 60.08206 Time: 242

=> Facility Event Loop Iteration #: 27
=> Facility<0> Name: f1 Product: p1 Result: 33.95246 Time: 260
=> Facility<1> Name: f2 Product: p2 Result: 63.84775 Time: 252

=> Facility Event Loop Iteration #: 29
=> Facility<0> Name: f1 Product: p1 Result: 35.06819 Time: 280
=> Facility<1> Name: f2 Product: p2 Result: 69.39277 Time: 272

=> Facility Event Loop Iteration #: 30
=> Facility<0> Name: f1 Product: p1 Result: 37.00819 Time: 290
=> Facility<1> Name: f2 Product: p2 Result: 75.23674 Time: 282
```

<표 9>는 특정 설비 federation의 실행에서 출력된 데이터의 일부를 나타내고 있다. 이곳에서는 federate에서 발생한 이벤트나 결과를 일정 시간 주기로 화면에 출력한다. "Facility Event Loop Iteration # 27" 경우 Facility[0]은 설비 이름이 "f1", 생산하는 제품의 종류는 "P1", 시간 260까지 약 33.9546개의 제품을 생산한 것을 나타낸다.

4.3.2 대기장소

<그림 11>은 대기장소의 모습을 표현하고 있으며, 각 생산 설비 federation에서 발생하는 상황을 볼 수 있는 모니터링의 역할을 겸하고 있다. 이 모니터에서는 일정 시간 동안 본 연구에서 RTI의 구현을 위해 개발된 프로그램의 수행 결과를 나타내고 있다. 이 그림은 대기장소 federation에서 모니터링을 수행할 때 역시 볼 수 있으며 각 federation의 수행 결과 및 대기장소에서 적절한 명령에 따라 이를 수행한 결과의 모습이다.

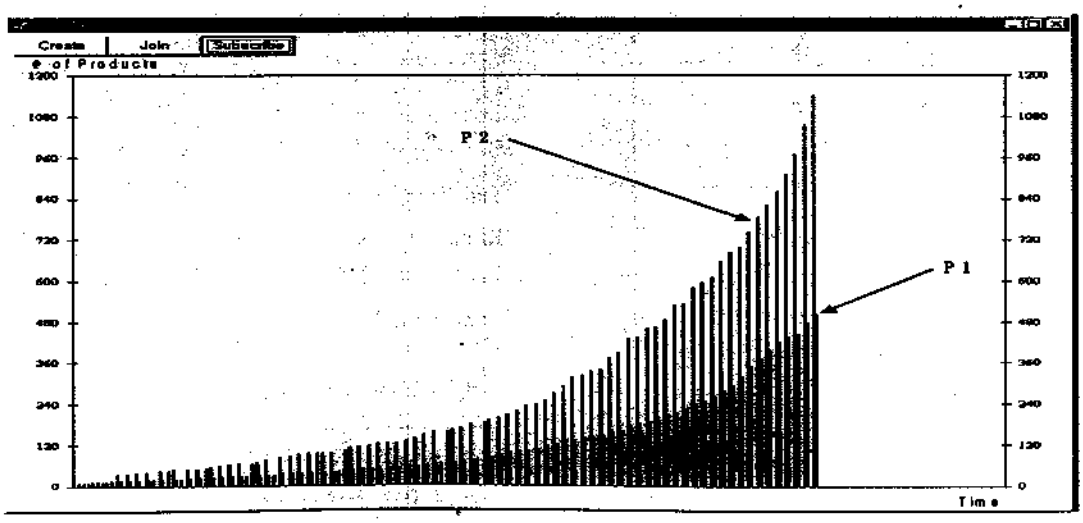

그림 11. 대기장소의 모니터링 상황.

5. 결 론

HLA는 기본적으로 DoD의 장기적인 M&S(Modeling and Simulation) 전략에 기본을 두고 있다. 1999년 이후 HLA가 아닌 simulation에 관한 재정지원을 하지 않고, 2001년 이후에는 HLA 호환이 아닌 시뮬레이션들은 은퇴를 시킨다는 방침이다. 즉 현재의 상호연동 표준인 DIS, ALSP 등을 대체하겠다는 의도이다. 관련 분야 종사자들의 연구, 개발에 대한 참여와 관심이 요구되고 있다.

HLA/RTI의 기본 목적은 분산 시스템 간의 상호작용, 즉 분산 시스템 간에 서로의 데이터를 주고받는 전송 규약이다. 서론에서 언급 한 바와 같이 HLA/RTI는 미국 국방성이 중심이 되어 군사분야의 많은 분야에 적용하여 상당한 효과를 발휘하고 있으며, 현재는 민간부분에의 적용을 위한 많은 노력을 기울이고 있다. 다시 말해서, HLA/RTI는 컴퓨터 네트워크를 이용한 분산시뮬레이션연구에서 이미 중요한 의미와 역할을 담당하고 있다.

대다수 첨단 관련분야의 시작이 그랬듯이 HLA/RTI 역시 군사적 목적을 위해 시도되었다. 현재 이 HLA/RTI를 이용하여 개발된 응용범위도 국방관련 시뮬레이션연구 프로젝트를 중심으로 진행되고 있다. 점차로 응용범위가 확산되면서 사회 전반적인 분야에 도입 및 적용이 가능한 것으로 보고되고 있다. 국방분야 이외에도 게임, 엔터테인먼트, 지상 및 항공교통 등 컴퓨터 네트워크를 기반으로 하는 분산시뮬레이션이나 분산시스템에 적용이 시도되고 있고 충분한 가능성이 엿보이고 있다.

본 연구에서는 지금까지 미국 국방성에서 비롯되어 현재 IEEE의 표준으로 진행되고 있는 HLA/RTI의 개념과 방법론을 소규모 특정 생산시스템 시뮬레이션에 적용하여 이에 대한 가능성을 검토하였다. 일반적으로 생산시스템에서 관리되는 데이터의 양은 상당히 규모가 커서 이를 관리하기 위해서는 효율성 높은 공통의 네트워크 라이브러리가 필요하다. 이 라이브러리 시스템 내 소위 federate이라고 일컫는 단위 시뮬레이션의 시작과 종료는 물론이고, 이들 시뮬레이션 간의 데이터 전송, 전송 데이터의 규모 통제 및 경로 관리, 시뮬레이션 간 시간 동기화 문제 등 제반 호환규격에 입각하여 운용되어야 할 필요성을 지니고 있다. RTI는 이러한 요구를 충분히 만족시킬 수 있도록 설계된 라이브러리이다.

DoD에서 아직은 모든 HLA/RTI와 관련된 정보와 application을 제공하고 있어서 쉽게 접근할 수 있다. 이를 적극 활용하면 빠른 시간 내에 산업체 및 공공기관의 여러 서비스분야에 적용이 확산될 수 있을 것으로 기대된다. 저자들은 이 연구를 진행하는 과정에서 주변에서 쉽게 찾아 볼 수 있는 특정 통합시스템의 원활하고 효율적인 운영을 도모하거나 경쟁력제고를 위하는 차원에서 고려될 수 있는 분산시스템연구에 HLA/RTI 도입 또는 적용은 문제해결을 위한 새로운 방법으로 자리매김하고 있음을 확인할 수 있었다.

참고문헌

안중호 (1994), *경영과 정보통신 기술*, 학현사.

Bagrodia, R. L., Chandy, K. M. and Misra, J. (1987), A message-based approach to discrete-event simulation, *IEEE Transactions on Software Engineering*, 13(6), 664-665.

Bill Garbacz, Brian Plamondon, Victor Colon (1999). Lesson Learned in the SOM and FOM Development Process for a Legacy Simulator, *Spring Simulation Interoperability Workshop*.

Charles Herring (1995), Frederick Hayes-Roth, *The DSSA Process -Applied to ADS architecture*, 28(2)

Nicol, D. M. and Fujimoto. R. M. (1994), Parallel Simulation Today, *Annals of Operations Research*, 53, 249-286

DMSO[1], *http://hla.dmso.mil.*

DMSO[2], *High Level Architecture Interface Specification.*

DMSO[3], *High Level Architecture Run-Time Infrastructure.*

DMSO[4] (1999), *High Level Architecture Run-Time Infrastructure Installation Guide RTI 1.3.* 6.

Fujimoto, R. M. and Nicol, D. M. (1992), State of the Art in Parallel Simulation, *Proceedings of 1992 Winter Simulation Conference*, 246-254

Fujimoto, R. M.,(1990), Parallel Discrete Event Simulation, *Communications of the ACM*, 33(10), 31-53

Jack Ogren (1997), *EAGLE and the High Level Architecture*, DMSO.

Jefferson, D. R. (1985), Virtual Time, *ACM Transactions on -Programming Language and Systems*, 7(3), 404-425.

Jim Sikora, Phil Coose (1995), *What in the World is ADS?* PHALANX, 28(2).

John A. Hamilton, Jr, *et al.* (1997), *Distributes Simulation*, CRC Press.

John W. Shockely, Kirk Parsons, Mark Morgenthaler (1999), Developing an HLA Virtual Command Post, *SIMULATION*, 73(4).

Ken Hunt, Dr. Judith Dahmann, Robert Lutz, Jack Sheehan (1997), *Planning for the Evaluation of Automated Tools in HLA*, DMSO.

Lin, Y. and Fishwick, P. A. (1996), Asynchronous Parallel Discrete Event Simulation, *IEEE Transactions on System*, Man and Cybernetics, Part A, 26(4), 397-412.

Marco Cantu (1999), Inside Secretes Delphi 4, SYBEX.

Mikel D. Petty, Piotr S. Windyga (1999), A High Level Architecture -based Medical Simulation System, *SIMULATION*, 73(4).

Misra, J. (1986), Distributed Discrete-Event Simulation, *ACM Computing Surveys*, 18(1), 39-65.

Pitch Kunskapsurveckling AB, *http://www.pitch.se/*

Sochats, K. and Williams, J. (1992), *The Networking and Communications Desk Reference*, Prentice-Hall, Carmel, IN

Todd H. Repass (1995), *DIS: An Evolving Modus Operandi for the Department of the Navy*, PHALANX, 28(2).

Terrain Modeling Project Office, *http://www.tmpo.nima.mil/guides/Glossary*

The AEgis Technologies Group, Inc. *http://www.aegistc.com/*

Thomas Schulze, Steffen Stanßburger, Ulrich Klein (1999), Migration of HLA into Civil Domains: Solution and Prototypes for Transportation Applications, *SIMULATION*, 73(4).

홍윤기
고려대학교 산업공학과 학사
USC, OR 석사
USC, 산업시스템공학 박사
California State University at Northbridge
경영과학과 조교수
현재: 한성대학교 산업시스템공학부 교수
관심분야: 시스템시뮬레이션 Stochastic Modeling, Wargame Analysis

권순종
한성대학교 산업공학과 학사
한성대학교 산업공학과 석사
현재: (주)리누소프트 소프트웨어 개발팀장
관심분야: ASP, 인터넷, 시뮬레이션, GIS