碩 士 學 位 論 文

指導敎授 洪 允 基

生産시스템 시뮬레이션을 爲한 High Level Architecture/ Run-Time Infrastructure의 適用

２０００年 ２月

漢城大學校 大學院

産 業 工 學 科

産 業 工 學 專 攻

權 純 宗

碩 士 學 位 論 文

指導敎授 洪 允 基

生産시스템 시뮬레이션을 爲한 High Level Architecture/ Run-Time Infrastructure의 適用

위 論文을 産業工學 碩士學位 論文으로 提出함

２０００年 ２月

漢城大學校 大學院

産 業 工 學 科

産 業 工 學 專 攻

權 純 宗

權純宗의 産業工學 碩士學位 論文을 認定함

２０００年 ２月

審査 委員長 印

審 査 委 員 印

審 査 委 員 印

제 목 목 차

표 목 차

그 림 목 차

제 1 장 서 론

시스템이란 그리스어의 Systema에서 유래된 말로 "특정한 목적을 가지고 이를 성취하기 위해 여러 구성인자가 서로 유기적으로 연결되어 목적 달성을 위해 상호 작용하는 것"을 말한다.[1] 최근 사회 산업구조가 복잡해지고, 그에 따른 응용 분야도 다양해짐에 따라서 시스템 내의 여러 구성인자 자체가 또 다른 하나의 하부 시스템을 구성하고 있다. 이와 같은 상황의 변화로 인해 특정 시스템을 컴퓨터 상에서 구현한다는 것은 매우 힘들다. 특히 시뮬레이션 분야에 있어서 최근의 산업시스템을 모델화하는 것은 거의 불가능할 수도 있다. 시스템의 특성상 서로 독립적으로 운영되기보다는 하나의 시스템 운영 결과가 다른 시스템의 입력 변수로 사용되는 경향이 늘어나고 있다. 이에 대한 대안으로서 제시된 개념이 분산 시뮬레이션이다. 원래 분산 시뮬레이션의 목적은 시뮬레이션 모델을 여러 컴퓨터에 나누어 실행하여 전체 시뮬레이션 수행 시간을 줄이는 것에 있다.

분산 시뮬레이션과 관련하여 발전하고 있는 분야 중에서 최근 대표적인 분야로 등장한 것이 컴퓨터 네트워크 분야이다. 컴퓨터 분야에서는 최근 수년 전부터 네트워크와 관련된 기술이 급속도로 발전되었다. 특히 인터넷과 관련된 분야는 개발자들이 충분히 숙지하여 응용할 만한 시간적 여유가 없을 만큼 빠르게 발전, 확대되고 있는 상황이다. 최근의 분산 시뮬레이션은 이와 같은 컴퓨터 네트워크 기술을 응용하여 개발되고 사용되고 있다. 분산 시뮬레이션에서 컴퓨터 네트워크가 차지하는 위치는 매우 핵심적인 분야가 되었고, 네트워크를 이용한 데이터의 교환은 필수적이다. 이에 따라서 분산된 시스템간의 네트워크 전송을 이용하기 위한 많은 방법이 소개되었다. 그러나 개발된 네트워크 전송 방법은 개발툴들 마다 모두 다르기 때문에 사용자 측면에서는 하나의 유일한 개발툴만 사용하여 시스템 및 분산 시뮬레이션 모듈을 개발할 수 밖에 없다.

현재까지 분산 시뮬레이션에 관한 연구는 많은 학자들에 의해 개발되고 새로운 방법들이 제시되었다.[10] 그러나 이들 대부분의 연구에서 발생되는 문제는 주로 시간의 동기화(Time Synchronization)와 이에 따른 시뮬레이션 모듈간의 데이터 전송에 관한 문제로 귀착되곤 하였다.[23] 특히 시간의 동기화에 관한 문제가 주를 이루었는데, 이에 관한 대안으로 제시된 것이 보수적 알고리듬(conservative algorithm)과 낙관적 알고리듬(optimistic algorithm)이다.

보수적 알고리듬은 시뮬레이션 모듈에서 이벤트를 진행시킬 때 안전한 이벤트라고 보증되는 이벤트만 진행하는 방식이다.[12] 반면에 낙관적 알고리듬은 이벤트 순서 변화를 염두에 두지 않고 각 시뮬레이션 모듈에서 독립적인 이벤트를 진행시키는데, 시뮬레이션 도중 이벤트의 순서 변화가 발생하면 모든 시뮬레이션 상태를 그 시간의 상황으로 되돌린 후 다시 시뮬레이션을 하는 방법이다[9]. 하지만, 이 두 가지 알고리듬이 모든 분산 시뮬레이션에서의 해결책을 제시하지는 못하였으며, 오히려 더욱 복잡한 결과를 가져오기도 하였다.

미국 국방성(Department of Defense; DoD)은 최근 새로운 개념의 분산 시뮬레이션 개념을 제안하고 있다.[5] DoD에서 제안한 분산 시뮬레이션의 개념은 HLA(High Level Architecture)라 불리는 것으로, 시뮬레이션 모델들 간의 상호 운용성(interoperability)을 높이기 위한 것이다.

일반적인 분산 시뮬레이션은 시뮬레이션 모델을 여러 컴퓨터에 나누어 실행하여 전체 시뮬레이션 수행 시간을 줄이는 방법이다. 지금까지는 여러 컴퓨터에 나뉘어 실행되는 시뮬레이션 모듈은 서로 동일한 시뮬레이션 언어로 구현된 모델을 사용하였으며, 시뮬레이션 모듈을 수행하는 운영체제 또한 동일하거나 유사한 운영체제를 사용하였다.

반면에, HLA는 분산된 시뮬레이션 모델이나 모듈들 간의 상호 운용성을 보장하기 위해 제안되었으며, 특히 Wargame이나 훈련 교육 등과 같이 시뮬레이션

모델과 인간이 참여하는 시뮬레이션(Human-in-the-loop)들 간의 상호작용을 할 수 있도록 뒷받침하는 개념을 포함하고 있다. 더욱이, 과거 분산 시뮬레이션에서 단점으로 지목되었던 이기종 시뮬레이터간 연동의 불가능에 대한 제약조건을 극복하였고, 시뮬레이션 개발 언어가 다르다 하더라도 상호작용이 가능하게끔 구성되었다.

또한 최근에는 이 HLA를 IEEE의 표준으로 제안하여 사용하기 위한 노력을 기울이고 있어, 머지않아 분산된 시스템간의 데이터 전송방식에 표준으로 자리잡게 될 것이다.

미국 국방성 산하의 DMSO를 비롯한 관련 학계에서는 HLA의 확장성 및 성능의 효과를 인지하여 많은 노력과 연구를 계속해왔다. 특히 1998년 초부터 SRI International and Lionhearth Technologies는 HLA 기술을 이용하여 Virtual Command Post (VCP)라는 것을 개발하기 시작하여 이미 실무적인 작업에 착수하고 있다.[15] VCP는 전장에서 사용자 즉 전투에 참가하는 구성원들 간의 상호작용과 전장에서 발생하는 각종 정보를 서로 공유하기 위한 분산 시뮬레이션이다. 또한 의학 시뮬레이션 분야에서는 훈련과 분석을 위해 이미 HLA와 기존의 의학 시뮬레이션을 통합한 어플리케이션을 개발하고 있다.[20] 이미 미국에서는 군사분야의 시뮬레이션에서 HLA는 필수적인 항목으로 되어 있고, 현재는 물자의 수송과 같은 민간분야의 시뮬레이션 등에도 HLA를 적용하려는 노력을 하고 있다.[27] 이들의 HLA에 대한 공통적인 결론은 분산 시뮬레이션에서 HLA는 네트워크의 표준으로 사용될 것이며, 학계와 산업 전반에 미치는 영향은 지대할 것으로 예측하였다. 특히 실제 시스템과 인간의 참여, 가상 환경 속에서 벌어지는 각종 시스템의 통합에서 핵심적인 역할을 수행할 것으로 기대된다.

위에서 언급한 HLA의 경우 발생 초기에는 ADS라는 개념이 주를 이루었다.

ADS는 시간적으로 응집되어 있고, 상호 작용 가능한 지역적으로 서로 분산되고 서로 다른 형태의 시뮬레이션을 지원하기 위한 기술로부터 시작되었다.[13] 그 이후 여러 지원분야의 기술을 통합하여 HLA라는 개념을 구현하게 된 것이다. 미국 국방성에서는 이미 수 년전부터 HLA 개념을 이용한 군사훈련과 각종 시뮬레이션을 수행하고 있다.[24][3]

본 연구에서는 위에서 언급한 HLA의 개념 및 방법론을 생산시스템에 적용하여 활용 가능성은 물론 기타 시스템으로의 확장성을 검토해 보고자 한다.

제 2 장 분산 시뮬레이션 기술

2.1 분산 시스템

2.1.1 분산 시스템

분산 시스템이란 사용자 인터페이스(user interface)를 사용하여, 최종 사용자(end user)에게 네트워크 통신을 통하여 서비스를 제공하기 위해 서버를 사용하는, 상호 협동하는 구성인자들의 단일 집합을 말한다.[14] 분산 시스템에서는 주로 네트워크를 이용하여 원격지에 있는 시스템간의 상호작용을 수행하며 원격지는 주로 지역적, 기능적 및 임무에 따라서 관련되어 구성된다.

2.1.2 분산 시스템 형태

분산 시스템은 일반적으로 다음에서 보는 것과 같이 주로 5가지 형태로 구성된다.[16]

· Distributed Operating System : 다중 처리장치(multiple processor)들이 고속의 네트워크 장비로 상호 연결되며, 기능성을 중심으로 분산된 운영체제를 말한다.

· Distributed Processing System : 서로 유사한 트랜잭션이 많은 곳에서 수행되거나 단일 트랜잭션이 다중 지역에서 처리되는, 네트워크 기반의 Computing system을 말한다.

· Distributed Application : 독립적인 방법으로 각 설비와 장비에 관리와 처리 기능을 할당하는, 프로세서의 구성과 기타 정보처리 및 네트워크 처리 장비를

말한다.

· Distributed Database : 데이터 베이스의 분할이나 복제를 위해 네트워크 내의 다중 지역에 저장되는 데이터 베이스를 말하며 위의 3가지와는 달리 저장 방식에 따른 분류이다.

· Distributed File System : 파일들이 네트워크 노드에 분산되어 있으나, 보조 기억장치에 저장되어 사용 가능하게 구성된 파일 시스템을 말하다.

2.2 분산 시뮬레이션

2.1.1 분산 시뮬레이션

분산 시뮬레이션은 시뮬레이션을 기능별, 임무별 등으로 나누어 서로 다른 프로세서에서 진행시켜 전체적으로 시뮬레이션 수행시간을 줄이는 시뮬레이션의 한 방법이다. 하나의 시스템에서 발생하는 부하를 다른 프로세서에서 진행하므로 그만큼의 시스템 운영 효율을 가져올 수 있는 것이다. 또한 각각의 특성에 맞게 서로 분산시켜, 차후 발생 가능한 유지 및 보수의 관점에서 이익을 가져올 수 있다.

과거에는 구현해야 하는 시스템의 구조가 복잡하지 않았고, 시뮬레이션의 기술 또한 단순했었다. 또한, 구현하기 힘든 모델에는 많은 가정을 두어 그 모델을 단순화 시켰다. 따라서, 구현된 시뮬레이션 모델들은 많은 가정으로 인해 현실성이 떨어지게 되었고, 출력결과의 신뢰도 또한 낮았다. 이에 대한 대안으로 제시된 것이 분산 시뮬레이션 방법론이다.

분산 시뮬레이션 방법론을 사용하여 시뮬레이션 모델을 구축하면, 시뮬레이션 수행 도중 과부하로 인한 시스템의 정지를 막을 수 있다. 또한, 기능별 또는 임

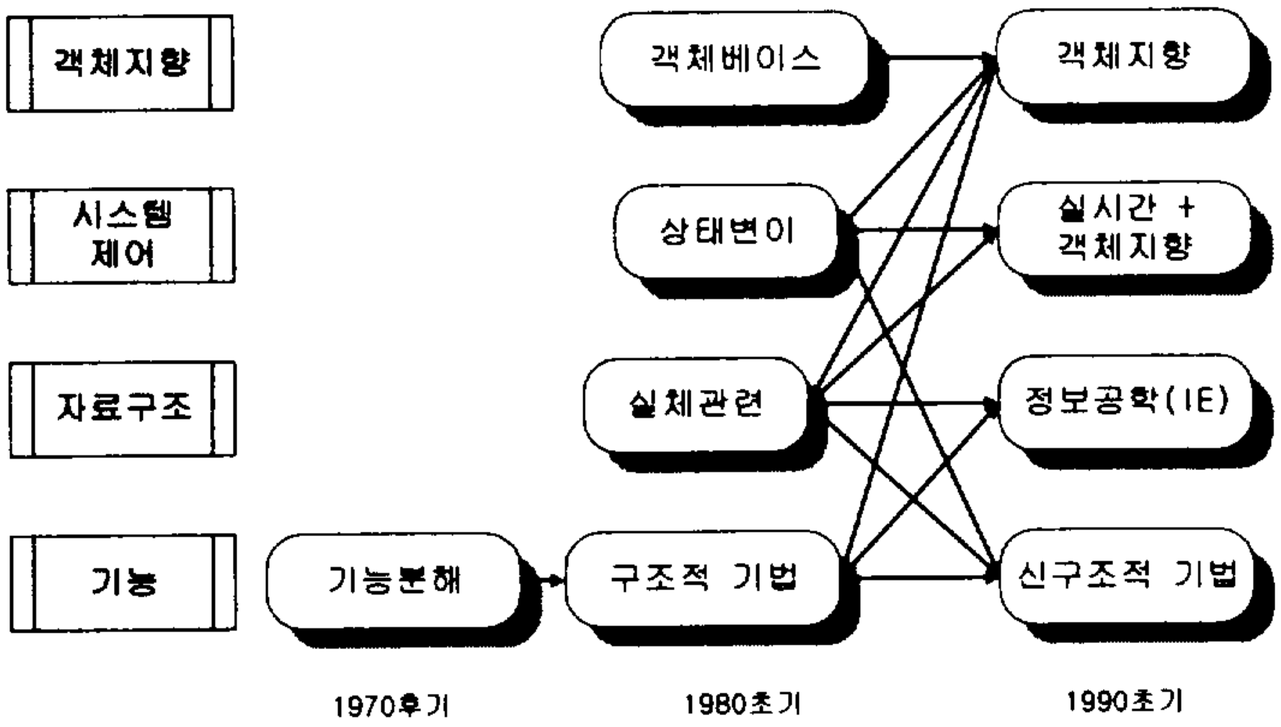
무별로 서로 분할된 프로세서에서 수행되므로 유지 및 보수의 측면에서도 많은 효과를 가져올 수 있다.

2.3 객체지향 모델링

2.3.1 객체지향 모델링

최근 흔히 객체지향이라는 용어를 자주 접하게 된다. 객체지향은 1980년대 말에 등장하였다. 그 전에는 주로 구조적 프로그래밍(structured programming)에 기반을 둔 구조적 접근(structured approach)이 프로그램 모델링 분야에서 많이 사용되었다. 그러나, 최근의 정보들이 방대해지고, 개방화되고, 인터넷으로 향하는 추세에 기존의 자료처리 방식으로는 도저히 현재의 수요를 감당해 낼 수 없을 뿐더러 이용하는 것조차도 한계에 부딪혔다. 이러한 소프트웨어의 위기 속에서 등장한 것이 객체지향의 개념이다.[19]

객체를 사용하는 이유는 우선 사용하기가 쉽고, 확장하기가 쉬우며, 유지 및 보수하기가 쉽기 때문이다. 객체지향 모델은 모듈(module)의 개념을 갖는다. 모듈이라는 것은 과거에 라이브러리라고 표현했던 것과 유사한 것이다. 그러나 근



<그림 1> 분석설계 기법의 추이

본적으로 다른 점은 이 모듈 안에 자료뿐만 아니라 기능(function)의 특성도 가지고 있다는 점이다.

객체지향은 일반적으로 추상화(abstraction), 캡슐화(encapsulation), 상속성(inheritance), 다형성(polymorphism)의 특성을 가지고 있다.

(1) 추상화

현실세계의 물체를 객체에 사상(mapping)할 때에는 현실의 그대로 객체로서 표현하는 것이 아니라, 문제의 중요한 측면 내지는 강조하고 싶은 내용을 표현한다. 객체에 의해 현실 세계를 추상화하는 효과는 3가지를 들 수 있다.

첫째, 객체에 대한 접근을 기능중심 보다는 물체에 중심을 둔다. 이는 변경에 대해 대응이 용이하기 때문에 안정을 줄 수 있다.

둘째, 과거에 얻어진 객체를 재 이용하기도 하고, 실 세계화와 비유를 이용해 필요한 기능과 행동을 유추할 수 있다. 이러한 특성은 실세계를 자연스럽게 표현할 수 있는 장점이 된다.

셋째, 현실세계의 중요한 측면만을 모델화하기 때문에, 분석시 초점이 명확해진다.

(2) 캡슐화

객체지향 접근은 자료와 절차를 그룹화한 객체를 단위(module)로서 소프트웨어를 개발한다. 이때, 단위(module)는 객체가 보유하고 있는 자료와 절차에 대해 외부에서 가시성(visibility)을 적절히 제공한다. 즉 객체 외부로부터 접근할 수 있는 것을 절차를 통해 간접적으로 자료를 접근하지만, 객체 자료에는 직접 접근할 수 없게 할 수 있다. 이 상태는 마치 자료가 캡슐로 둘러싸여진 상태에 비유할 수 있기 때문에 이것을 '캡슐화(encapsulation)' 혹은 '정보은닉(information hiding)'이라고 한다.

캡슐화의 효과는 사용자 측면과 개발자의 측면을 나눌 수 있다. 사용자의 측면에서 본 캡슐화의 효과는 사용자가 일일이 자료의 구조를 알 필요가 없다는 것이다. 단지 자료 전달에 필요한 모수(parameter)들만 알고 있으면 되므로 개발에 신속한 결과를 가져올 수 있다. 반면에 개발자의 측면에서는 내부 자료구조를 변경해도 사용자에게 영향을 주지 않는다는 장점이 있다. 시스템 내부의 자료구조가 변경되었다해도 다른 사용자들에게 매번 전달해 줄 필요가 없다는 것이다.

(3) 상속성

기존의 시스템과 언어에서는 개개의 구성요소 부 모듈을 그대로의 형으로 재이용할 수 있었지만, 그 일부를 다소 변경한다든지 일부 기능을 추가하는 것은 불가능하다. 그 때문에 기존의 것과 약간의 다른 것을 필요로 할 경우에도 처음부터 다시 개발하는 중복개발이 필요했다. 그러나 객체지향 접근은 상위 클래스(class)의 개념을 하위 클래스에 상속시킴과 동시에 하위 클래스에는 상위 클래스에 존재하지 않는 새로운 성질을 추가할 수 있다. 즉 상속(inheritance)이라는 메커니즘을 소개하고 있는 것이다. 상속성의 특징으로 인해 기존의 자원을 유효하게 재 사용해 생산성을 향상시킬 방법을 제공하고 있다.

상속성은 일단 개발된 모듈을 체계화할 수 있도록 한다. 이와 같은 특징은 개발자 측면에서 볼 때 모듈의 검색이 쉽다는 장점이 있다. 또한 상위 클래스에서 파생된 하위 클래스의 이용에서 볼 때는 모듈의 확장과 변경이 쉽다는 효과를 가져올 수 있는 것이다.

(4) 다형성

다형성은 상관관계가 있지만 서로 다른 2개의 객체가 동일한 명령을 해석(interpret)한다. 하지만 그것들 나름대로의 방식으로 다르게 동작할 수 있는 능력을 의미한다.

2.3.2 구조적 프로그래밍 기법

객체지향은 1980년대 말경에 등장하였다. 80년대는 구조적 프로그래밍(structured programming)에 기반으로 발표한 구조적 접근(structured approach)이 세상에 바람을 일으켰던 시기이다. 개발의 대상의 기능에 중점을 두는 구조적 방법은 일괄처리 방식인 자료 변환을 중심으로 응용 소프트웨어의 개발에 유용하게 사용되었다. 그러나, 시대가 변하고 프로그래밍에서 관리해야 하는 자료의 수와 종류가 급격하게 증가함으로 인해 이와 같은 구조적 접근에서는 문제점이 발생하게 되었다. 우선, 기능(function)이 컴퓨터 상에서의 실행 단위인 것과 같이, 분석 및 설계 단계에서도 기능 중심의 문제 접근이 일관적으로 사용되었다. 이와 같은, 현실에서의 문제는 기능보다는 물체의 구조로서 적용하는 방법이 자연스럽다. 분석과 설계단계에서 물체를 중심으로 접근할 수 없었던 것은, 접근하더라도 절차형 언어로 구현하기 어려웠기 때문이다.

두 번째로, 기존의 절차형 언어에 있어서는 자료에 대한 접근의 제한은 아무것도 없었기 때문에, 개발자가 자유롭게 자료의 참조나 갱신을 수행할 수 있었다. 그러나, 이러한 자유로움으로 인해 개발자는 자료를 접근할 때 자료에 대한 모든 구조를 이해해야만 했다.

세 번째로 드러난 문제점은 중복 개발이다. 기존의 언어에서는, 기존의 절차나 자료 구조의 확장은 대단히 손이 많이 가는 작업이었다. 코드의 일부를 변경하는 것과 같은 확장의 경우, 우선 코드 전체를 복사하고, 복사된 코드에 대해서 수정, 추가를 덧붙인다. 이와 같은 작업으로 인해 유사한 코드를 가준 복수의 절차나 자료구조의 양이 상당히 많아지게 되었다. 게다가, 기존의 코드에 에러가 있으면, 복사 및 확장된 코드에도 동일한 에러를 포함하고 있으며, 수정시 모든 코드에 대해 에러 수정을 가해야 하는 힘든 작업이 필요하게 되었다. 이러한 문제들을 해결하기 위해 등장한 것이 객체 지향적 접근(Object Oriented Approach)이다.

2.3.3 객체지향 모델링의 장점 및 효과

객체 지향의 가장 큰 특징은 현실의 물체를 주목하는 것이다. 실세계로부터의 자연스러운 발상으로 시스템을 구성한다. 이러한 관점은 바로 상향식 (bottom-up) 개발에 알맞다. 기능을 포함한 물체를 하나의 객체로 표현하므로, 추후 개발시에는 표현된 객체를 조합 및 구성하는 방법을 사용하여 하나의 시스템을 구축하게 된다. 이러한 특징으로 인해서 요구 정의 명세변경에 유연하게 대응할 수 있고, 소프트웨어 개발의 산출물을 재사용할 수 있게 되었다. 또한 그룹 개발 즉, 분산 개발을 원활하고 신속하게 할 수 있으며, 지적 자원의 재활용을 가능하게 한다.

이와 같은 객체 지향의 장점과 효과로 인해 근래에 개발되는 모든 프로그램들과 시스템들은 객체 지향의 방법을 이용해서 개발되고 있는 실정이다.

2.4 DIS와 ADS

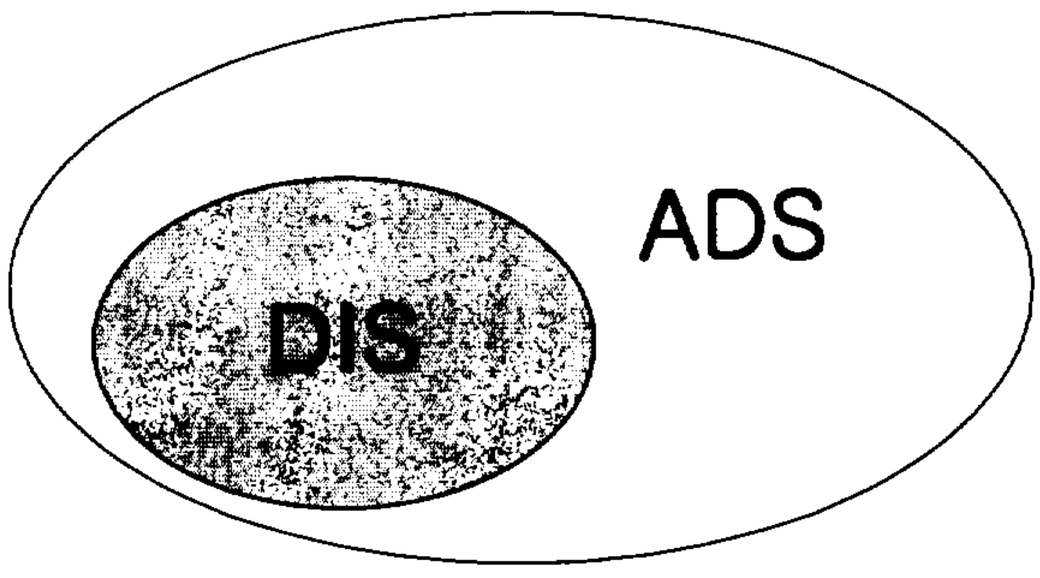
DoD(Department of Defense)는 정보시스템을 요구하는 곳에 정확하고 적절하게 제공하기 위해 항상 신속한 정보시스템을 전개시켜왔다. 초기의 목적은 훈련의 응용 분야에 초점을 맞추었고, Simulation Network(SIMNET) 프로그램으로부터 전개하였다. SIMNET은 ARPA(Advanced Research Projects Agency)와 육군의 STRICOM(Simulation, Training, and Instrumentation Command)에서 관리되었다. 개념적으로, 이 프로젝트는 공통의 가상 환경 속에서, 지역적으로 분산된 지역에서, 오퍼레이터가 있는 훈련 시뮬레이터를 연결하는 방향으로 관리되었다. 이때, 오퍼레이터들은 거의 실시간(near-real-time)으로 이 공통의 환경에서 서로 상호작용 할 수 있었다.

수년 동안 SIMNET은 좀 더 많은 유연성과 강건하고, 그리고 서로 다른 단계

의 충실도(fidelity)로 기술 개선시켜왔다. 이러한 기술을 DIS(Distributed Interactive Simulation)이라고 부른다. DIS의 첫 번째 임무는 다중 지역에 분포되어 있는 여러 가지 형태의 시뮬레이터들을 서로 연결하는 기반 구조를 정의하는 것이었다. 이 기반 구조는 각각의 목적들을 위해, 다른 시대로부터의 기술들, 여러 공급업자들이 납품한 제품들, 여러 서비스의 플랫폼들 사이에 상호 작용을 위해 구축되었다.

DIS 기반구조는 종합적인 환경 내에서 인터페이스 표준, 통신 구조, 관리 구조, 기술적 포럼 및 이기종 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.

DIS의 중요한 영역은 시뮬레이션과 종합적 환경에서 human-in-the-loop의 상호작용이다. 그러나, 시뮬레이션 영역에서, DIS는 시간 조절이나 요구되는 데이터 전송에 맞추지 못하는 경우가 발생하기도 하였다. 예를 들어, 충실도가 높은 기술적 응용과 같은 분야에서 발생되는 문제는 시간과 개념적 문제를 포함하고 있다. 이러한 것들은 종종 테스트와 평가(test and evaluation)의 영역에서 발생한다.



<그림 2> DIS와 ADS의 관계

ADS(Advanced Distributed Simulation)는 IEEE 1278에 기술된 표준을 넘어서는 네트워크의 구현을 포함하고 있다. ADS는 DIS를 부분 집합으로 포함하며, Live, Virtual, Constructive simulation 요소의 혼합을 지원한다. Live simulation은 실제 사람이 실제의 장비나 도구를 사용하는 시뮬레이션을 말한다. Virtual simulation은 실제 장비가 아닌 3차원 가상 환경으로 구성된 시뮬레이터에서 실제 사람이 행하는 시뮬레이션을 말한다. 반면에 Constructive simulation은 일반적으로 말해지는 시뮬레이션이다. 다시 말해서, 컴퓨터가 만들어내는 각종 환경과 상태 하에서 수행되는 시뮬레이션이다.

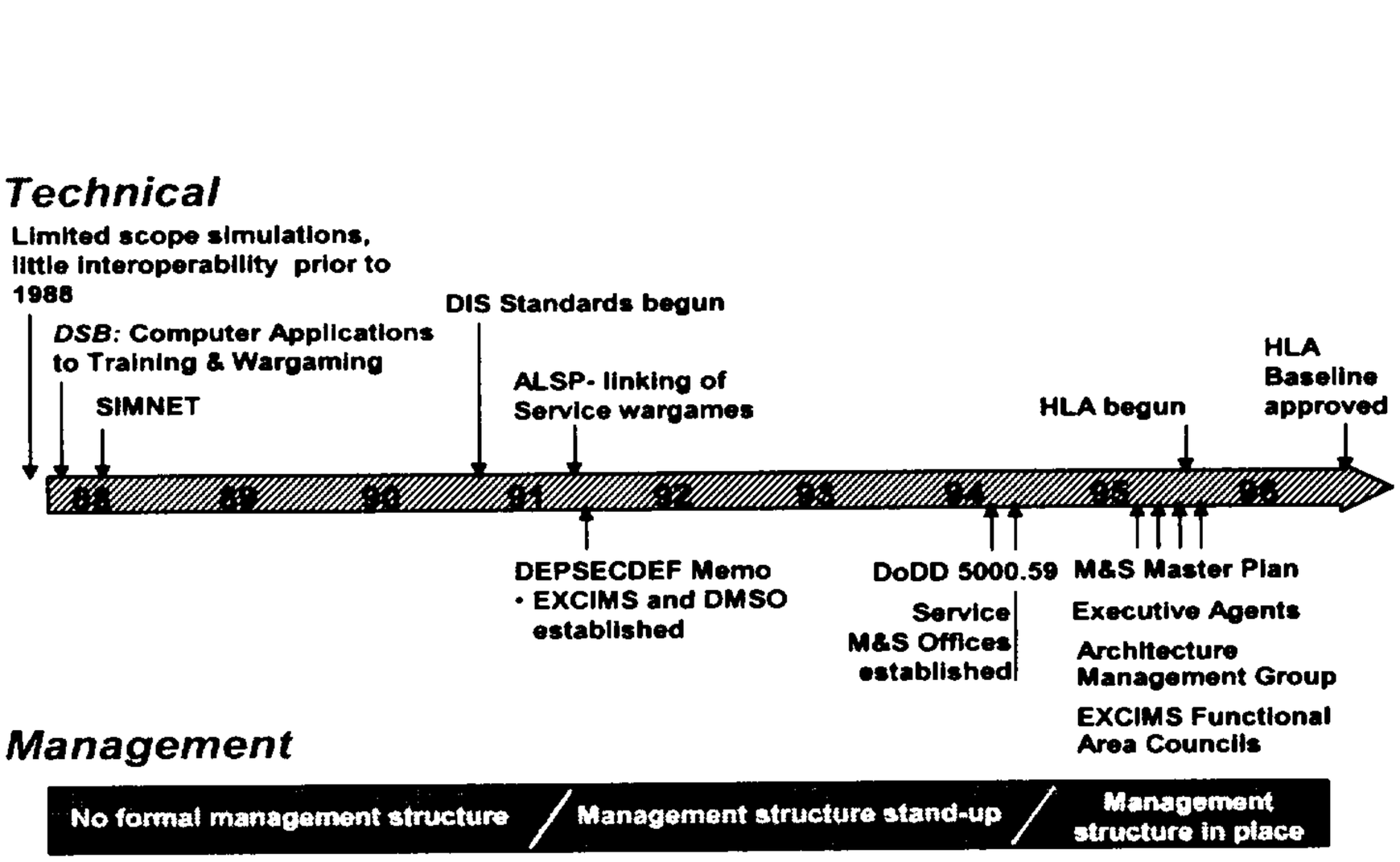
2.4.1 DIS 정의

DIS는 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 혹은 그 이상의 지역적으로 분산된 곳에 위치될 수도 있다.[6][24]

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

2.4.2 ADS의 정의

ADS는 DIS에서 단점으로 지적된 서로 다른 적용 범위에 있는 시뮬레이션들 간의 데이터 전송과 시간의 동기화를 맞추기 위해 등장했다. ADS는 시간 응집성, 종합적 환경 하에서의 상호작용을 제공하기 위해 나타난 기술이다.[24] 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, constructive 시뮬레이션이 혼합된 시뮬레이션에서의 상호작용도 지원하게 되었다. ADS 개념의 핵심을 이루는 것이 다음에 언급될 HLA이다. <그림 3>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 하게되었다.



<그림 3> DIS부터 HLA까지의 발전 과정

2.5 High Level Architecture (HLA)

2.5.1 HLA 정의

간단히 말해서, HLA는 "시뮬레이션의 재사용과 상호작용을 위한 아키텍처"라고 말한다.[5]

대부분의 첨단 관련 학문의 시초가 국방과 관련한 문제에서 비롯되었듯이 발생되었듯이, 미국 국방성(DoD)을 중심으로 HLA가 나타나게 되었다. 원래 HLA는 미국 국방성에서 ADS를 구현하기 위한 하나의 아키텍처로 시작되었다.

HLA 개념의 적용분야는 일반적인 사회 모든 분야에 적용될 수 있다. 국방 분야는 물론이고, 게임, 엔터테인먼트, 교통, 생산 등 다른 객체들의 정보를 참조해야 정확한 결과를 도출할 수 있는 모든 분산 시스템 분야에 적용이 가능하며, 현재 많은 활동이 이루어지고 있다.

HLA의 일반적인 사용으로 인해, 운영과 조달 분야에서 각각의 주어진 임무를 수행하기 위한 긴밀한 상호작용이 활성화 될 것이다. 또한, 유용성과 유연성을 최대화하기 위해 이러한 모델링과 시뮬레이션 환경은 오픈 시스템 아키텍처를 통한 상호 작용성을 적절히 사용하며, 재사용 가능한 컴포넌트로부터 구축될 것이다. 하나의 완전무결한 컴포넌트의 구축은 추가적인 재개발의 필요성을 제거해 줌으로써, 결과적으로 운영에 소요되는 각종 자원의 사용에 효율을 높일 수 있을 것이다.

HLA의 좀 더 세밀한 정의는 다음과 같다.

"모든 DoD 시뮬레이션에 적합한 중요한 기능적 요소, 인터페이스 및 설계 규칙과 특정 시스템 구조에서 일반적인 기반구조를 제공하기 위해 정의된다."

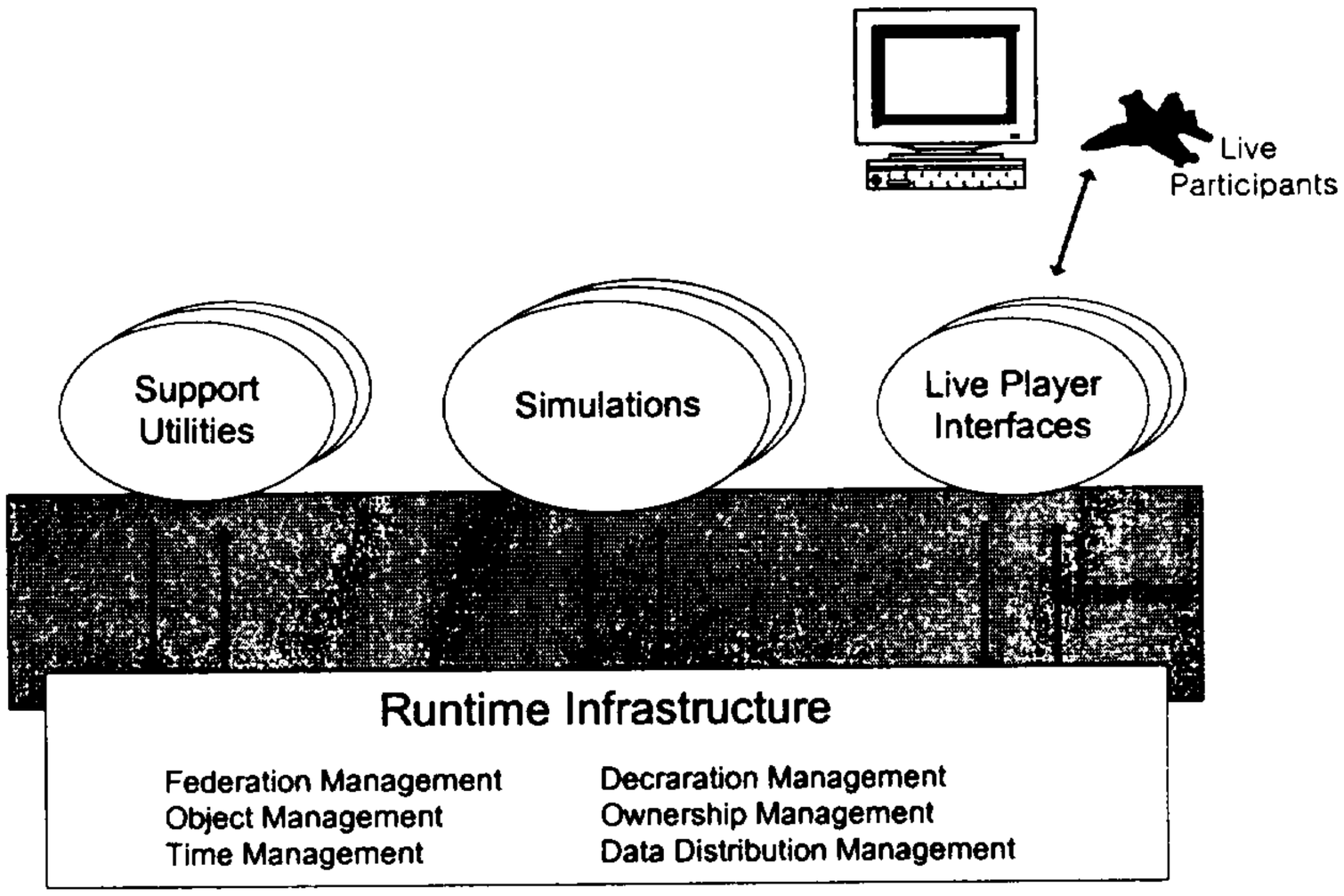
앞에서도 언급되었지만, HLA는 한 순간에 나타난 것이 아니다. 과거 국방 시뮬레이션(SIMNET)에서, 지역적으로 떨어진 시뮬레이터들 사이에 상호작용의 취

약점을 보완하기 위해 DIS가 등장하였으며, 이 DIS에 데이터 전송과 시간 동기화 등의 기능을 추가시켜 개선한 것이 ADS이다. 바로 이 ADS를 구현하는 아키텍처가 HLA인 것이다. 따라서 HLA에는 위에서 말한 데이터 전송 문제와 전체 시뮬레이터간의 시간 동기화를 위한 기능들이 강조되었으며 기능적으로 강화되었다. 이 부분은 RTI(Run-Time Infrastructure)를 이용하여 구현하게 된다.

<그림 4>는 각각의 특성을 지닌 시뮬레이션 모듈과 RTI의 관계를 아키텍처의 기능적 관점에서 바라본 그림이다. 각각의 시뮬레이션 모듈은 특정의 interface를 통하여 RTI와 상호작용하는 것이다.

2.5.2 구성요소

일반적으로 HLA는 HLA Rule, Interface Specification 및 Object Modeling Template의 3가지 구성요소를 가지고 있다.



<그림 4> 아키텍처의 기능적 관점

1) HLA Rule

Federation이라고 불리는 시뮬레이션들의 더 나은 상호작용을 위해 반드시 따라야 하는 규칙들이다. 여기에는 시뮬레이션들의 책임과 HLA federation들의 RTI(Run-Time infrastructure)를 포함하고 있다. 10가지의 규칙으로 구성되어 있다. 이곳에 기술된 내용을 따르지 않으면, 추후 다른 federation들과의 상호작용이 힘들며 자체 federation의 확장 및 유지, 보수에 어려움을 나타내게 된다.

2) Interface Specification

RTI와 시뮬레이션 오브젝트 사이의 인터페이스 기능을 정의하고 있다.

3) Object Modeling Template

각각의 federation과 시뮬레이션을 위해 HLA 오브젝트 모델에서 요구되는 정보를 기록하기 위한, 미리 정의된 일반적인 방법이다. 다시 말해, 이곳에는 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.

2.6 Run-Time Infrastructure (RTI)

HLA는 소프트웨어가 아니라 하나의 아키텍처이다. 반면에, RTI(Run-Time

infrastructure)는 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. 현재 DMSO(Defense Modeling and Simulation Office)에서 주관이 되어 RTI를 개발하고 배포하고 있다.

<표 1> HLA Interface Specification

구성요소	기능
Federation Management	· Create and delete federation executions · Join and resign federation executions · Control checkpoint, pause, resume, restart
Declaration Management	· Establish intent to publish and subscribe to object attributes and interactions
Object Management	· Create and delete object instances · Control attribute and interaction publication · Create and delete object reflections
Owner Management	· Transfer ownership of object attributes
Time Management	· Coordinate the advance of logical time and its relationship to realtime
Data Distribution Management	· Supports efficient routing of data

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 플렛폼하에서 사용할 수 있도록 제공하고 있다.

2.6.1 구성요소

RTI는 <표 1>에 표시된 것과 같이 크게 6가지 구성요소로 구성되어 있으며 각각의 Management에서 전체 RTI의 운영에 필요한 기능들을 수행하고 있다. Federation Management에서는 시뮬레이션 모듈이 RTI와 연계해서 작동할 수 있는 기본 환경의 생성과 소멸 등에 대한 내용으로 구성되어 있으며, Declaration Management에서는 다른 시뮬레이션 모듈들과 상호작용을 할 수 있는 기반 구조를 정의한다. Object Management에서는 전체적으로 RTI와 시뮬레이션 모듈이 서로 상호작용을 할 때 사용되는 각종 객체(object)의 생성과 그에 따른 속성들을 정의하게 해 주며, Owner Management는 객체 속성의 소유권을 다른 곳으로 변경할 수 있는 기능을 제공한다. Time Management에서는 각 시뮬레이션 모듈들 간의 시간적 동기화와 관련된 기능을 포함하고 있다. 마지막으로 Data Distribution Management에서는 실제적인 시뮬레이션 모듈들 사이에서 데이터의 전송과 수신등에 관한 기능을 수행한다.

2.6.2 RTI에서 사용되는 용어

1) Federation: 시뮬레이션과 공통 FOM(federation object model) 및 RTI의 집합을 말하며 큰 규모의 모델이나 시뮬레이션의 형태로 사용된다.

2) Federate: federation의 구성요소로 하나의 시뮬레이션을 말한다.

- federate는 전투 시뮬레이션과 같은 하나의 환경을 표현할 수 있다.

- federate는 항공 교통 흐름의 국가적 시뮬레이션과 같이 통합되어 표현될 수도 있다.

3) Federation Execution: federation의 실행

4) Object: federation에 의해 시뮬레이션되는 영역 내의 객체이며,

- 하나 이상의 객체로 구성된다.
- Run-Time Infrastructure에 의해 관리된다.

5) Interaction: 임시적이며, 하나의 federate에 의해 발생된 시간 기록의 이벤트와 RTI를 통해 다른 것으로부터 전달받는 것.

6) Attribute: FOM에서 정의된 자료로서, 오브젝트의 클래스의 인스턴스와 관련이 있다.

7) Parameter: FOM에서 정의된 자료로서, 클래스 상호작용의 인스턴스와 관련이 있다.

제 3 장 생산시스템 모델 설계

3.1 생산시스템 구현 목적

3.1.1 생산시스템에서의 분산 시뮬레이션의 필요성

많은 생산 현장에서 실제 생산계획과 더불어 많은 생산 관련 시뮬레이션을 병행하여 시행하고 있다. 그러나 대부분의 생산에 있어서 많은 변수로 인해 정확한 시뮬레이션 결과를 도출하기는 어렵다. 생산시스템이 복잡해지고 다양화됨으로써 더욱 시뮬레이션 결과를 생산계획에 반영하는 데에는 한계성을 실감하게 된다.

최근 생산분야 뿐만 아니라 기타 산업분야에서 시스템 운영 효율을 높이기 위해 시스템을 모듈화하여 분리하여 운영을 시도하고 있다. 그러나, 특정 시스템의 운영을 위해서는 모듈화된 시스템들 사이에 서로 데이터의 교환이 원활히 이루어져야 한다. 다시 말해서 다른 곳에서 발생된 시뮬레이션 결과가 또 다른 시뮬레이션에 중요한 입력변수가 될 가능성이 다분히 존재하기 때문이다. 특히 생산분야에 있어서는 생산시스템 형태상 대부분 이전 생산 모듈에서 발생한 출력이 다음 생산 모듈의 입력요소가 된다. 이러한 방식으로 많은 생산 모듈들 사이에 원활한 상호작용이 이루어지는 것이다.

현재의 생산시스템에 분산 시스템의 개념을 도입한 사례는 주변에 다수 존재한다. 그러나, 이들 모델에서는 시스템 자체의 목적보다는 데이터 전송과 시간을 관리해 주는 기능의 규모가 더 커지는 것과 같은 비효율성이 발생하기도 한다. 데이터 전송을 관리하는 적절한 규칙이 현재 다수 존재하지만, 각각의 업체별로 독자적인 모델을 구축하므로 이들 상호간의 상호작용의 측면에서도 부적절하다.

만일, 분산 시스템 및 분산 시뮬레이션의 개발시 네트워크 부분을 담당하는 공통의 라이브러리를 가지고 개발을 한다면 개발시간 뿐만 아니라 효율적 측면에서 상당한 효율을 얻을 수 있을 것이다. RTI는 이러한 공통의 네트워크 라이브

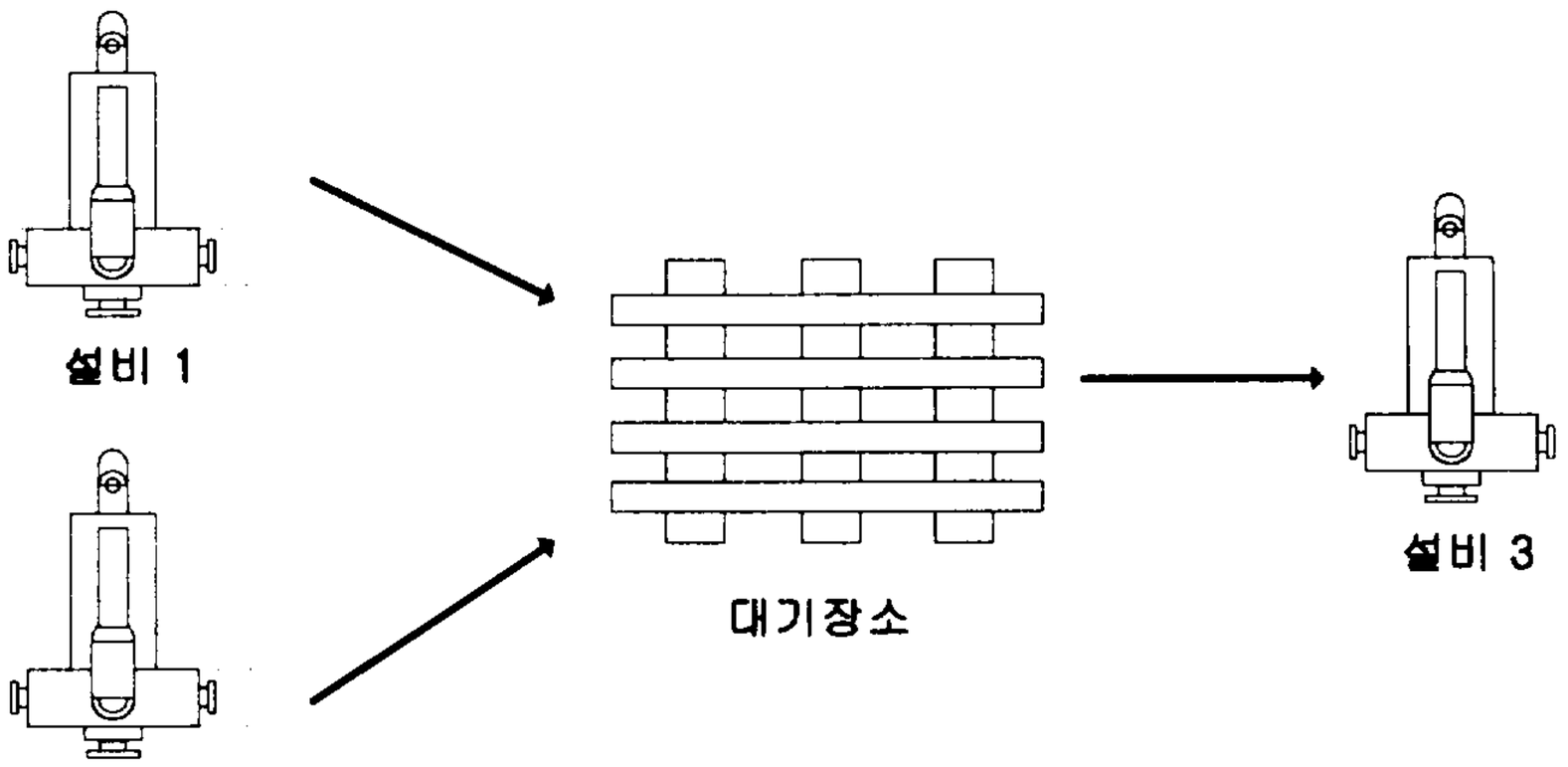
러리를 제공한다. 개발자는 이러한 기능을 가지고 있는 RTI를 객체지향적인 특성을 사용하여 손쉽게 사용할 수 있다.

3.2 생산시스템 구조

3.2.1 적용 대상 생산시스템의 구조

본 연구에서 사용된 생산시스템 모델은 생산시스템에 RTI 적용 가능성의 모색을 그 목적으로 하기 때문에, 시스템의 복잡성을 최소화하는 지극히 단순한 생산시스템을 그 모델로 한다. RTI는 여러가지 기능을 포함하고 있으나, 가장 핵심적인 기능은 바로 분산된 시스템 혹은 federation들간의 상호작용이다. 따라서 본 연구에서 사용된 모델은 federation들간의 상호작용에 그 연구중심을 맞추었다.

본 연구에서는 <그림 5>와 같은 형태의 생산시스템을 가정하였다. 각각의 생산 활동과 관련된 기계 및 모듈은 이론적으로 개수의 제한이 없으며, 시뮬레이션 실행 도중에도 참여 및 이탈이 가능하다. 각각의 설비에서는 자체적으로 생산율을 가지고 있으며, 대기장소의 경우 각 생산된 제품을 소유하게될 정해진 공간을



<그림 5> 생산시스템 모델

포함하고 있다. 만일 대기장소의 선행 설비에서 생산한 제품을 대기장소에서 수용할 수 없을 경우 대기장소는 선행 설비에 이 사실을 통보하여 생산을 멈추게 된다.

3.2.2 OMT 구축

Object Model Template는 앞에서 설명한 바와 같이, 각 federate들이 상호작용을 통해 정보를 교환할 때 참조하는 하나의 reference의 의미를 가진다. 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.

OMT를 개발하는 방법은 여러 종류가 있고, 개발을 도와주는 소프트웨어 역시 여러 개발사들에 의해 제공된다. 본 연구에서 사용하는 OMT 개발 소프트웨어는 AEgis Research Corporation에서 개발하여 HLA의 공식적인 OMT 개발툴로 되어 있는 OMDT(Object Model Development Tool)을 사용하였다. 추가적으로 Pitch Kunskapsutveckling AB사에서 개발한 Visual OMT를 덧붙여 사용하였다.[22][26]

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

기본적으로 사용되는 객체는 생산에 직접적으로 참여하는 설비객체이다. 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 보여지는 Object Class Structure Table은 전체 시뮬레이션에서 사용하는 객체들의 구조를 나타내며, 이것을 바탕으로 하여 federation execution내에서 객체를 생성한다.

1) 설비 객체 속성(Facility Object Attribute)

<표 2>에는 본 연구의 대상이 되는 모델에서 설비에 대한 속성을 나타내고 있다. "Name"은 해당 설비의 이름, "Product"는 해당 설비에서 생산 가능한 제품의 종류를 기록하고 있으며, "Result"는 생산 설비에서 일정시간동안 생산한 제품의 수를 담고 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> 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

<표 3>은 생산된 제품을 보관하게 될 대기장소의 속성을 표시한다. "Name" 항목은 대기장소의 이름이며, "P1_Count", "P2_Count"는 선행 설비에서 생산된 각각의 제품에 대해 대기장소에서 보관하고 있는 개수를 저장하게 된다.

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.

1) Publishable

특정의 object class는 RTI의 Publish Object Class 서비스를 이용하여 federate에 의해 참여한 다른 federate에 속성을 전달할 수 있게 한다. 또한 federate는 class의 이름을 사용한 RTI의 Register Object 서비스를 이용하게 된다. 다시 말해, publish는 자신의 정보를 다른 federation에 전송할 수 있는 능력을 말한다.

2) Subscribable

federate는 특정 class에서 object의 정보를 잠재적으로 재사용하거나 활용할 능력이 있다. 다른 object의 정보를 활용하는 능력은 이 class의 Discover Object의 RTI 메시지에 적절히 반영된 정보를 상황에 맞게 분류하여 사용하게 된다. 다시 말해, subscribe는 다른 federation의 정보를 수신할 수 있게 해 주는 능력을 말한다.

Publish는 RegisterObjectClass라는 함수를 사용하여 RTI에 자신이 전송하게 될 데이터 및 속성의 구조를 등록하며, Subscribe는 DiscoverObjectClass를 사용하여 등록된 데이터 및 속성의 구조를 알게되며, federation Execution에 참여

한 다른 federation을 참조할 수 있게 된다.

3.2.3 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)를 사용한다. 그러나 인터넷과 같은 TCP/IP를 사용하는 네트워크 환경에서도 무리 없는 진행이 가능하도록 RTI는 구성되어 있다.

제 4 장 생산시스템 구현

4.1 개발 환경

4.1.1 개발툴

윈도우 환경 하에서의 개발툴로는 마이크로소프트사의 Visual C++이 많이 사용되나, 개발시 소요되는 시간적 이유와 사용상의 어려움 및 난이도로 인하여, 최근 마이크로소프트사의 Visual Basic과 볼랜드사의 Delphi가 많이 사용되고 있는 추세이다. 이들 개발툴은 각기 OCX와 Component의 형태 및 Active X 컨트롤이라는 형태로 객체지향 방법론을 지원하고 있다.

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.

4.1.2 Active X

마이크로소프트의 비주얼 베이직(Visual Basic)은 소프트웨어 컴포넌트를 일반에 공급하는 개념을 소개한 최초의 프로그램 개발 환경이었다. 그러나 소프트웨어 컴포넌트 재사용의 개념은 앞에서 언급한 객체지향 방법론에 그 뿌리를 두고 있다. 이러한 객체지향 언어들은 재사용성을 전혀 제공해주지 못했는데, 그 이유는 상업화 문제나 표준화 문제에 기인한 것이라고 보아야 할 것이다. 비주얼 베이직은 객체지향 프로그래밍을 완전하게 구현해 내지 못하고 있지만, 프로그램 개발자들이 환경에 통합할 수 있는 새로운 컨트롤을 만들고 배포하는 표준적인

방법의 정의를 통해 컴포넌트의 개념을 제시하고 있다.

비주얼 베이직에 의해 처음으로 알려진 기술 표준은 VBX로서 16비트 체계이며, 32비트 플랫폼으로 전환하면서, 마이크로소프트는 VBX 표준을 좀더 강력하고 좀더 개방적인 Active X 컨트롤로 대체하였다. 최근에 등장하는 거의 모든 비주얼 개발툴은 이런 객체지향의 개념을 포함하고 있는 Active X를 사용할 수 있게 하였다.

4.1.3 개발된 Application

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 네트워크상에서 분리가 되어있어야 한다. 실험 환경에서는 각 federation은 모두 다른 컴퓨터에 설치가 되었으며, 네트워크로는 TCP/IP를 사용하였다. 네트워크의 사용은 LAN을 사용해도 상관없으나, 지역적인 제한이 없음을 입증하기 위해 인터넷망으로 많이 사용되고 있는 TCP/IP를 사용하였다.

본 모델에서 개발된 생산에 참여하는 객체는 크게 2가지 종류이다. 하나는 실제적으로 제품을 생산하는 설비이며, 다른 하나는 생산된 제품으로 보유하게될 대기장소이다.

4.2.1 생산 설비

앞의 OMT 개발 부분에서 언급했듯이 생산설비는 각각의 생산 가능한 제품에 대한 정보 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 생산 설비로 이 사실을 알려주게 된다. 또한 대기장소 다음에 위치하는 설비에서 생산이 가능하도록 제품을 전송하여 준다.

4.2.3 출력변수

출력변수의 감시는 주로 모니터를 수행하는 application에서 알게된다. 모니터에서는 전체적인 federation execution에 참여한 각각의 federation에서 벌어지는 상황을 감시한다. 상호 인지하는 출력변수는 다음과 같다.

1) 각각의 생산설비의 상태
2) 대기장소의 현재 규모
3) 생산설비에서 현재 생산하는 제품의 종류

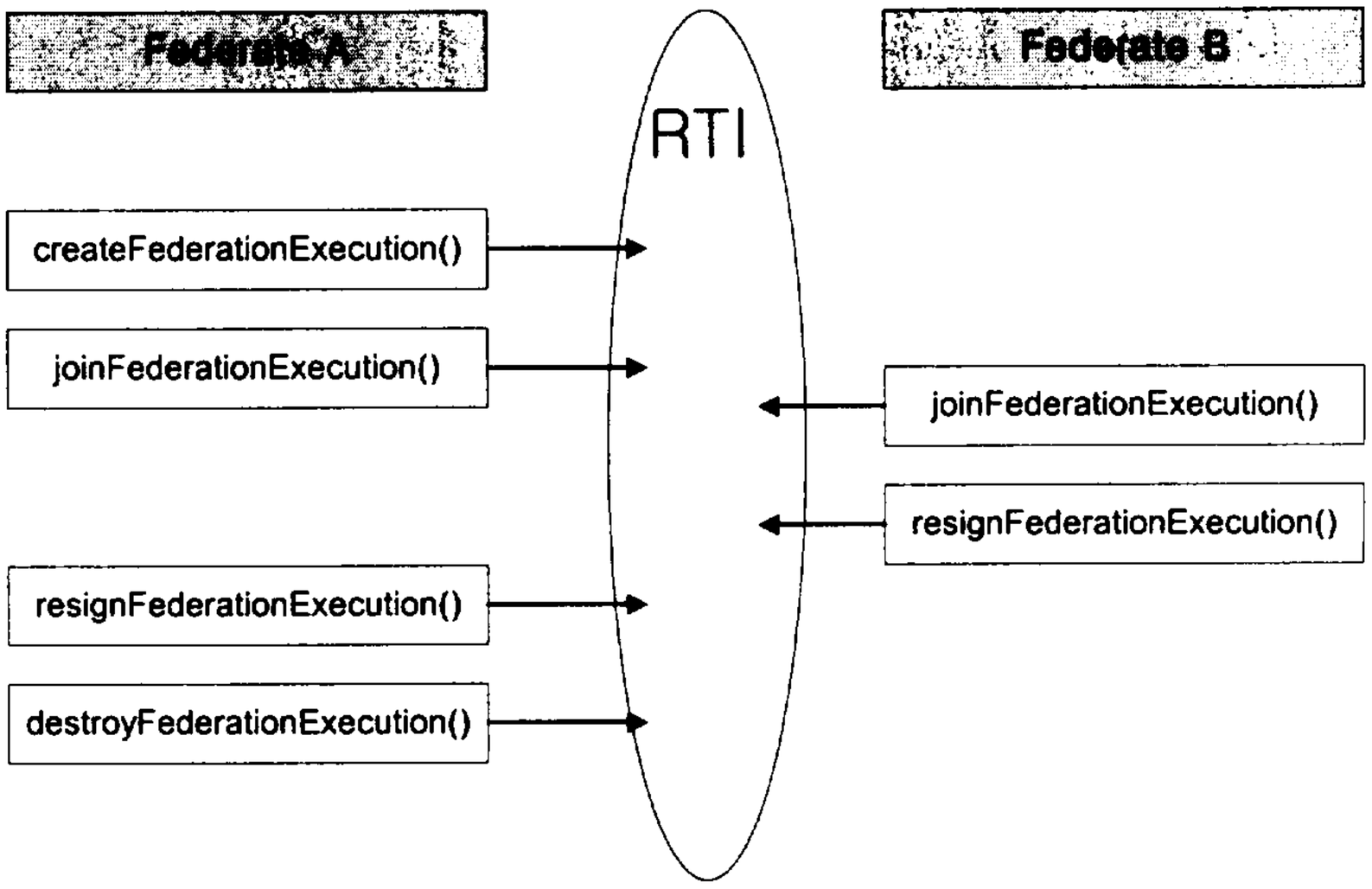
4.2.4 분산 시뮬레이션 개발시 RTI 도입 절차

1) RTI 환경 초기화 및 접속

우선 기본적인 시뮬레이션을 모듈별 및 기능별로 구분하는 것이 필요하다. 이

렇게 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.exe파일을 포함하고 있다. rtiexec는 RTI Executive를 말하는 것으

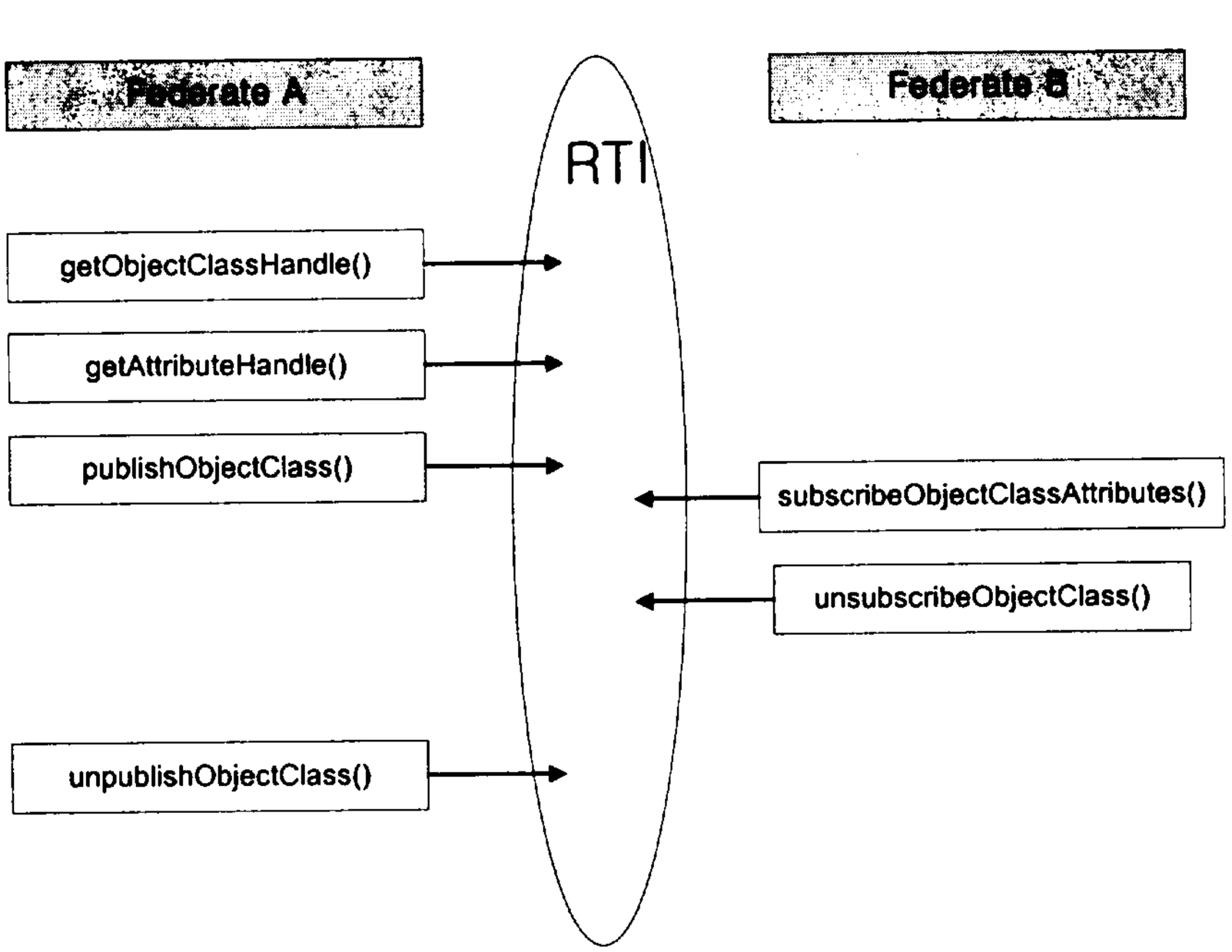


<그림 6> Federation Management Life Cycle

<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 );
```

로, 하나의 플렛폼에서 실행되며, 일반적으로 잘 알려진 Network Port를 사용한다. 또한 다른 이름으로 구성되는 여러 개의 federation execution을 관리하며 임의의 사용자 작업을 지원한다. 각각의 federation을 rtiexec를 통해 기본 object의 생성과 상호작용에 요구되는 object의 생성을 준비하게 된다. 일단 rtiexec가 실행된 후 <그림 6>에서 보는 바와 같이 federation은 CreateFederationExecution을 이용하여 federation에 맞는 환경을 구축한다. 이때 발생되는 공통의 환경을 fedexec라고 부른다. fedexec는 Federation Executive를 달리 부르는 말로, 실행 중인 federation당 하나의 프로세스를 발생하며, 여러 개의 federate를 관리한다. rtiexec는 유일하지만 이 속에서 발생할 수 있는 fedexec는 여러 가지가 될 수 있다. CreateFederationExecution에서는 미리 개발된 OMT와 이미 설정된 Federation Execution Name을 기준으로 하여 공통의 환경을 마련하게 되는 것이다. 정상적인 FederationExecution의 환경이 만들어지면, 만들어진 FederationExecution에 연결된다. 연결의 기능은



<그림 7> Object Publication and Subscription

<표 5> Publishable과 Subscribable 속성을 위한 예제

```
ms_facilityTypeId   =
            ms_rtiAmb->getObjectClassHandle(ms_facilityTypeStr);
ms_nameTypeId = ms_rtiAmb->getAttributeHandle(
                                ms_nameTypeStr, ms_facilityTypeId);
ms_productTypeId = ms_rtiAmb->getAttributeHandle(
                                ms_productTypeStr, ms_facilityTypeId);
ms_resultTypeId = ms_rtiAmb->getAttributeHandle(
                                ms_resultTypeStr,   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);
```

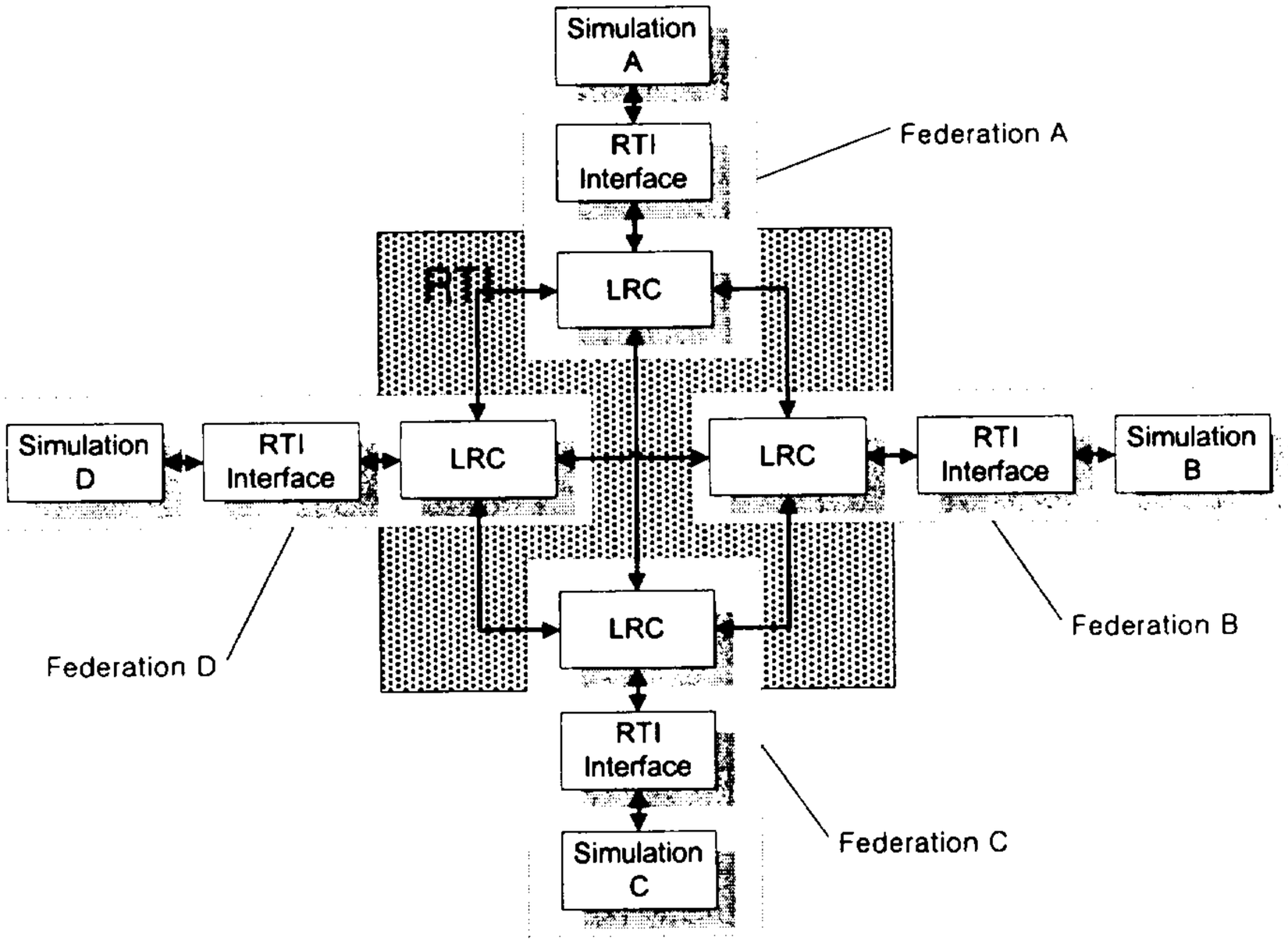
JoinFederationExecution이 수행한다. JoinFederationExecution은 위에서 생성된 FederationExecution의 이름과 자신의 Federation Name을 이용해서 연결하게 된다. <그림 7>에서는 연결시에는 추후 사용하게 될 OMT에서 기술된 각 Object와 Attribute에 해당하는 ID를 부여받는 개념을 나타낸 그림이다. 이때 부여받은 ID를 기준으로 하여 다른 federation과의 상호작용을 하게 된다. 추가적으로 Interaction class에 대한 handle도 부여받는데, 이는 객체에 대한 상호작

용 이외의 기타 메시지 전송 등과 같은 상호작용에 사용하게 된다.

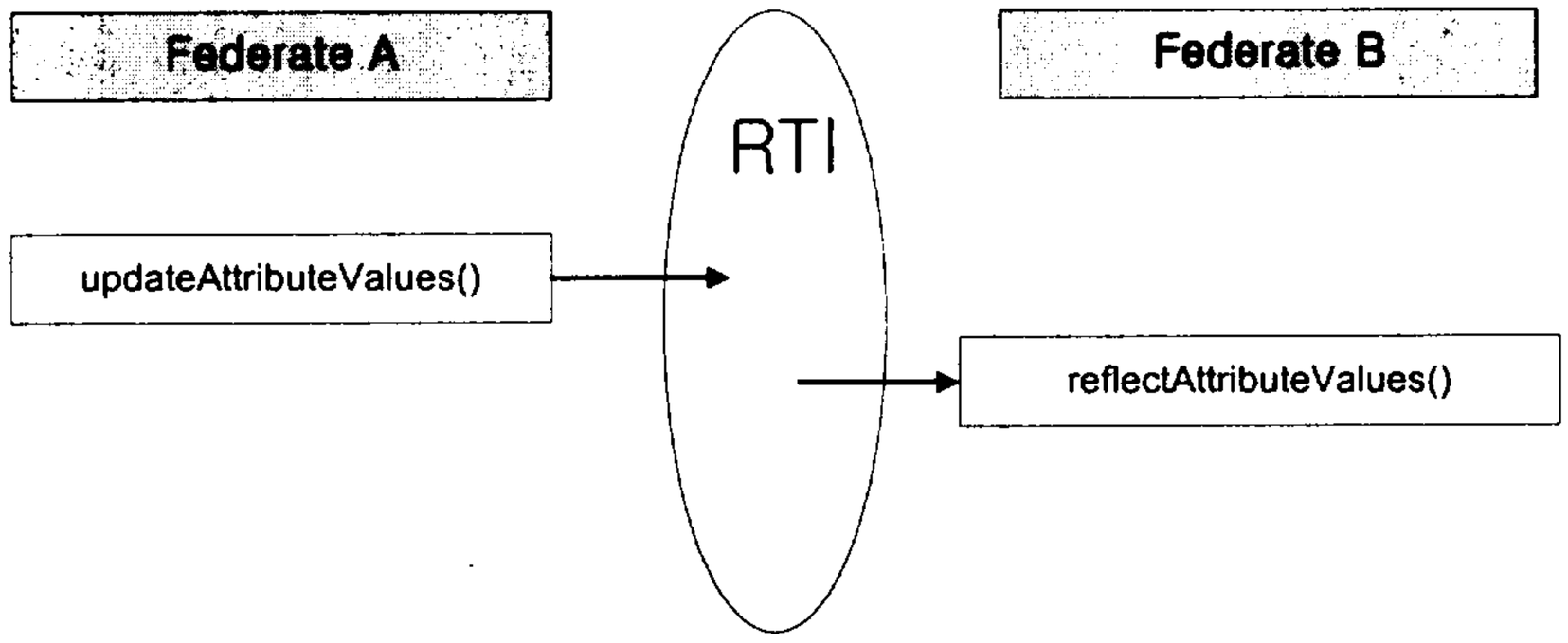
<표 4>와 <표 5>는 위에서 언급된 여러 가지 함수들이 실제로 어떠한 방식으로 사용되는 지를 표시한 것이다.

2) 데이터 전송을 위한 오브젝트의 속성 설정

<그림 7>은 정상적으로 federation이 만들어지고 만들어진 federation에 접속한 후 특정 federation은 시뮬레이션 수행동안 상호작용에서의 역할을 federation execution으로 발표하는 과정을 나타낸 그림이다. <그림 9>는 위에서 언급한 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



<그림 8> Simulation/RTI Interface Module을 포함한 Federation의 예



<그림 9> 속성 데이터의 변경시 전송과 수신

덧붙여 subscribe에서는 자신이 수신하게 될 object의 종류 및 attribute의 크기를 가지고 SubscribeObjectClassAttributes를 사용하여 마찬가지로 federation execution에 등록한다.

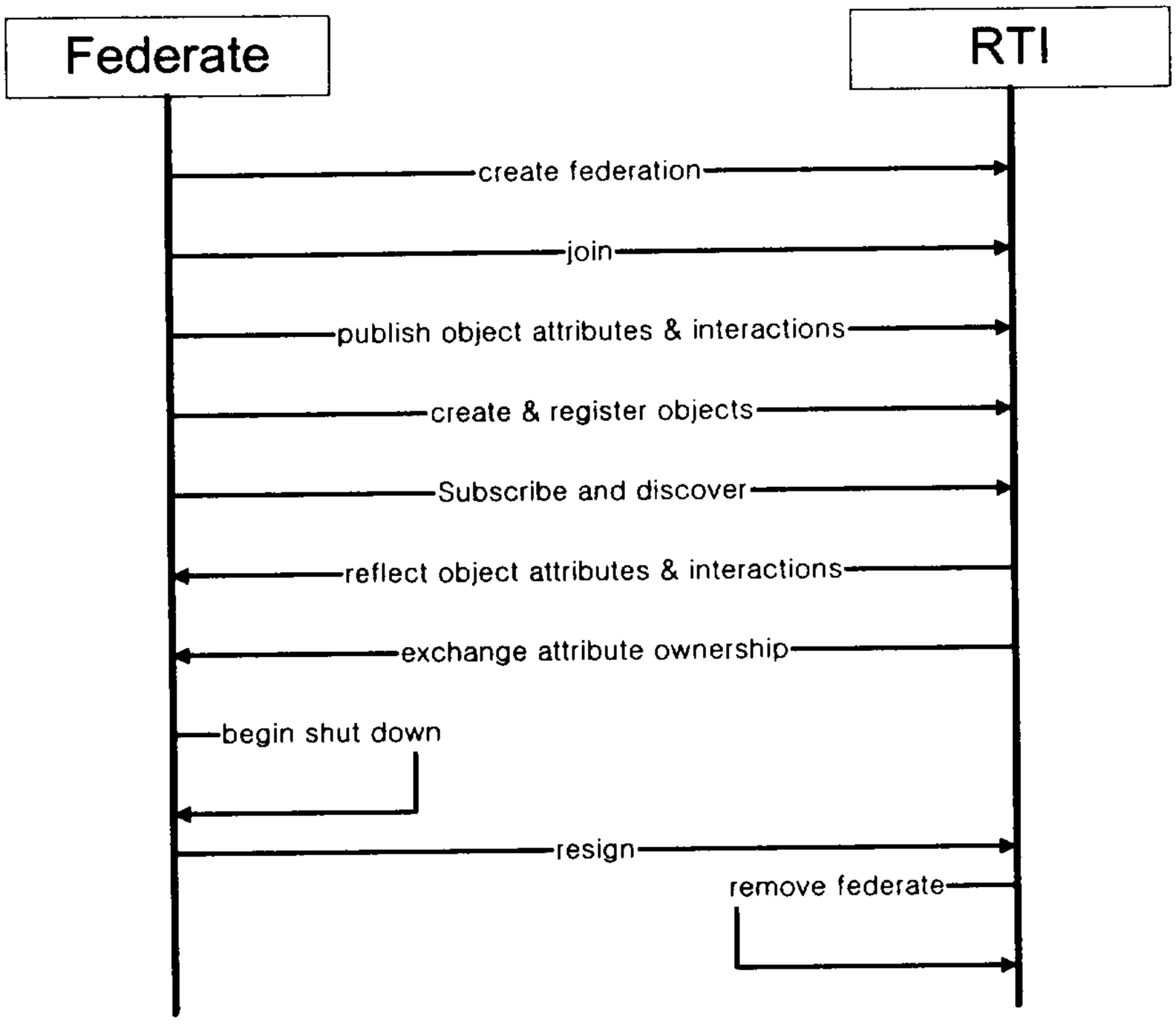
Publish와 Subscribe가 완료되면, 특히 Subscribe가 완료된 후 다른 federation에서 등록(publish)하게되면 자동적으로 subscribe를 등록한 federation으로 참가여부의 정보가 전송된다. 이때 작용하는 것이 DiscoverObjectInstance이다. 이 기능에서는 다른 federation이 등록했을 때, 그 때 사용한 객체의 이름, 객체 클래스 ID 및 객체의 Instance ID를 수신한다.

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

지금까지의 과정이 완료되면 <그림 8>과 같은 구조가 생성된다.

3) 데이터 전송

시뮬레이션 수행 중 다른 federation으로 공포해야할 메시지나 object의 attribute의 변경이 발생하면, *UpdateAttributeValues*의 기능을 이용한다. 이 기능은 attributeHandleValuePairSet과 자신이 가지고 있는 Object Handle을 이용



<그림 10> Federate - Federation Interplay

하여 다른 federtion으로 정보를 전달하는 임무를 수행한다. AttributeHandleValuePairSet에는 자신이 전송할 세부 Attribute의 값을 가지게 된다. 일반적인 메시지의 전송시에는 ParameterHandleValuePairSet을 사용하게 된다. 물론, 다른 federation으로 메시지나 Object의 값을 전송하기 위해서는 앞에서 언급한 publish의 기능이 수행되어 있어야 한다.

반면에 데이터의 수신측에서는 미리 subscribe의 기능이 수행되어 있어야 하

<표 6> 데이터의 전송과 수신을 위한 예제

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

고, 다른 federation에서 데이터의 전송이 있으면 *ReflectAttributeValues*의 기능을 수행하여 전송받은 데이터를 분류한다. 수신된 데이터에는 발신한 federation의 정보와 그곳에서 보낸 각 object 및 attribute의 정보를 포함하고 있다.

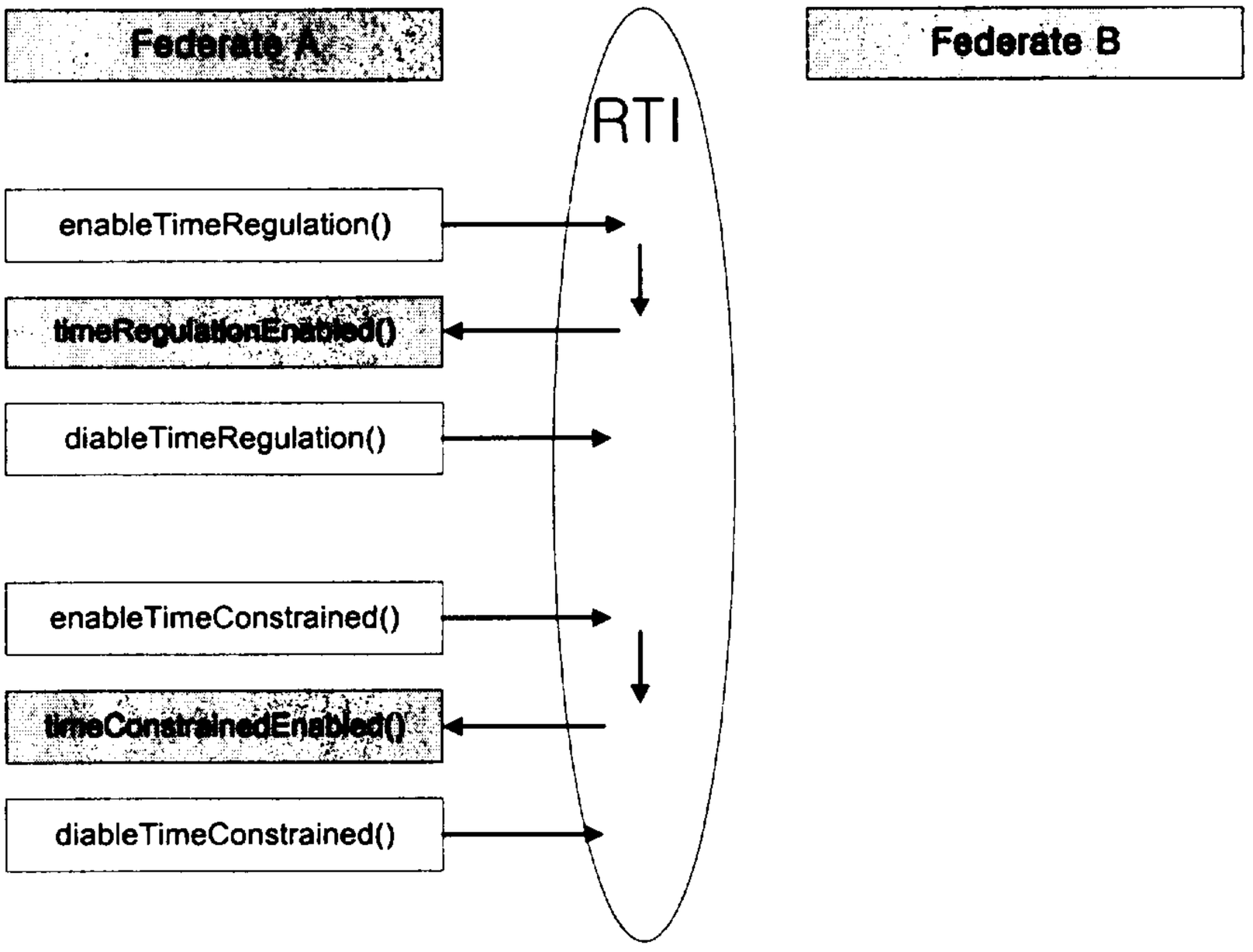
<표 6>은 데이터를 전송하기 위한 단계와 수신 후 원하는 정보를 추출하기 위한 과정의 예시이다.

일반적인 메시지의 수신시에는 ReceiveInteraction이 수행된다.

지금까지 설명한 federation의 생성부터 소멸까지의 과정이 <그림 10>에 나타나 있다.

4) 시간동기화

지금까지 설명한 것이 object attribute의 송수신과 일반적인 interacting message의 송수신에 관한 것이다.



<그림 11> "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 RTI 에서는 Time Management 항목에서 처리한다. 그 중에서 특히 강조되는 항목으로는 Regulating과 Constrained이다. <그림 11>은 Regulating과 Contrained의 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.

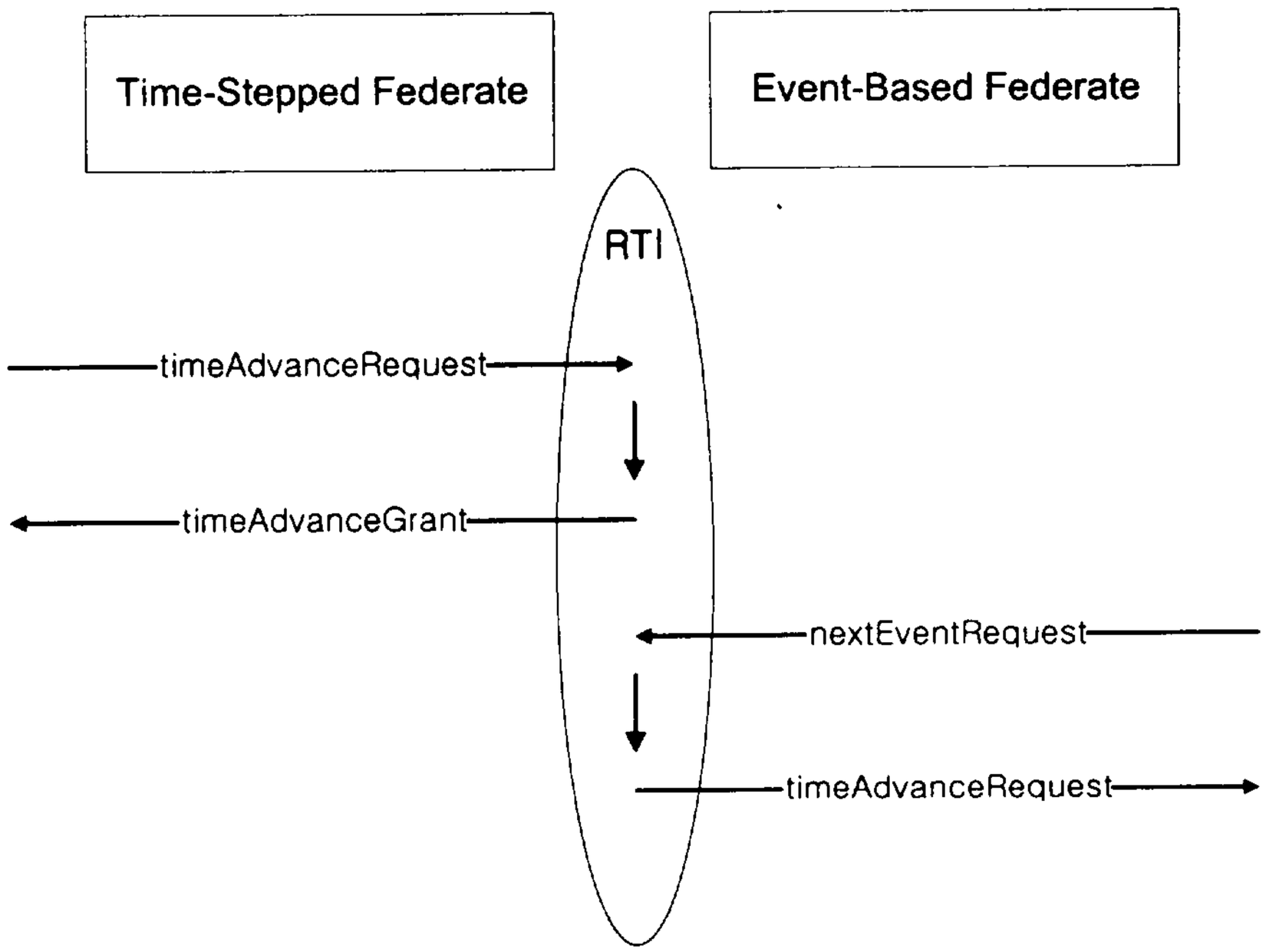
시뮬레이션에서 사용되는 시간의 진행에는 여러 가지 종류가 있다. 가장 많이 사용되는 방식은 Time-Stepped Simulation과 Event-Based Simulation 방식이

<표 7> Time Regulating과 Time Constrained를 위한 예제

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

다.

Time-Stepped 방식은 미리 설정한 시간 단위씩 시뮬레이션 시간이 증가되는 방식이다. Time-Stepped 방식을 사용하는 federate에서는 timeAdvanceRequest나 timeAdvanceRequestAvailable의 기능을 사용하여 RTI에서 시간 진행의 허가를 얻는다. timeAdvanceRequest나 혹은 timeAdvanceRequestAvailable을 전송하면 timeAdvanceGrant를 수신하여 다음 시간으로 진행한다. <표 8>은 Time-Stepped 방식에서 사용되는 RTI의 예제



<그림 12> Time-Stepped Federate와 Event-Based Federate에서의 시간진행과 관련된 함수

<표 8> Time Stepped Simulation에서의 시간진행 방법 예제

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

이다.

반면에 Event-Based 방식으로 진행되는 federate에서는 nextEventRequest나 혹은 nextEventRequestAvailable을 사용하여 마찬가지로 timeAdvanceGrant를 수신한다. <그림 12>는 각각 Time-Stepped Federate와 Event-Based Federate에서의 시간 진행 방식에서의 차이점을 나타낸 것이다.

4.3 생산 시뮬레이션 수행 결과

4.3.1 설비

기본적인 생산을 담당하는 application이다. 이곳에서 하는 일은 생산 가능한 제품을 event시간에 생산을 하고 대기장소로 정보를 전달하거나 대기장소에서의

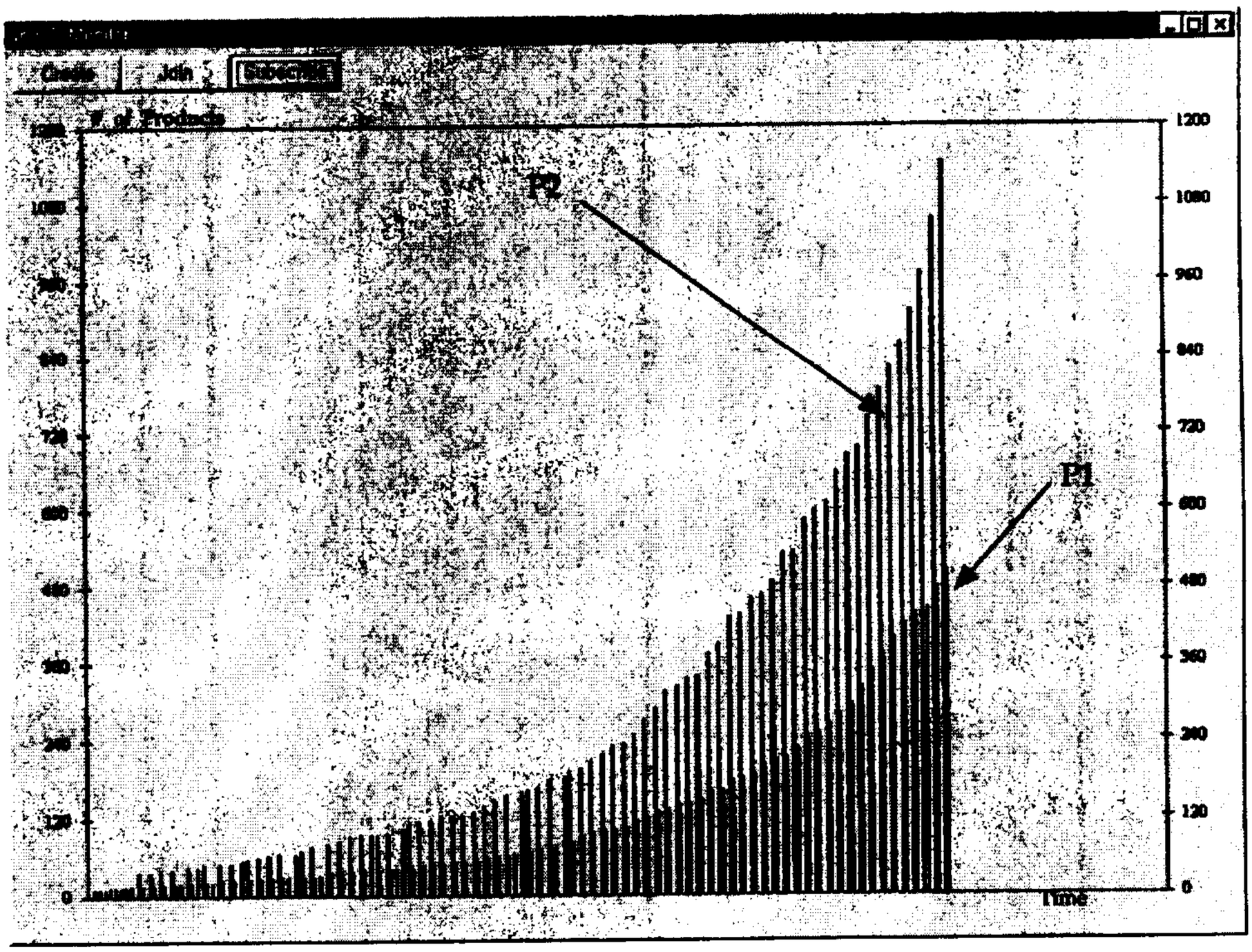
<표 9> Federation 실행결과

```
=> 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
```

정보를 수신한다. <표 9>는 특정 설비 federation의 실행에서 출력된 데이터의 일부를 나타내고 있다. 이곳에서는 federate에서 발생한 이벤트나 결과를 일정 시간 주기로 화면에 출력한다. <표 9>의 결과에서 "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

선행 설비에서 생산된 제품을 보관하는 역할을 하며 대기장소 다음에 위치한 설비에서 제품의 요구가 있을 경우 제품의 정보를 전송한다. 또한 대기장소에는 자체적으로 제품을 보관할 수 있는 용량의 한계가 있어서 이 한계에 도달하면 선행 설비로 정보를 보내어 생산을 중단시킨다. <그림 13>은 대기장소의 모습을



<그림 13> 대기장소에서의 모니터링 상황

표현하고 있으며, 각 생산 설비 federation에서 발생하는 상황을 볼 수 있는 모니터링의 역할을 겸하고 있다.

이 모니터에서는 일정 시간동안 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.

제 5 장 결 론

DIS는 최근 10여년간의 연구를 통하여 현재 그 효용성이 입증된 단계이다. 이에 반해 ADS를 구성하는 핵심적인 요소인 HLA는 DIS에서의 문제점들을 개선하고 분산 시뮬레이션 전반의 문제를 정의하기 위하여 새로이 설계되고 있는 중이다. HLA는 DIS에 비하여 다음과 같은 차이점들이 있다.

우선 DIS는 실시간만을 고려하지만 ADS에서의 HLA는 logical time을 포함한 복수의 시간 관리 스킴을 제공한다. DIS는 아키텍처에 데이터를 포함시켜 프로토콜의 확장이 제한되었지만 HLA는 이를 분리하여 응용의 요구에 따라 데이터를 변화시켜갈 수 있다. 또한 DIS는 full broadcast distribution approach이지만 HLA는 시뮬레이터간에 선택적으로 데이터를 전송하게 해준다. 이러한 차이점 외에 가장 큰 차이점은 HLA는 FOM등의 라이브러리를 통하여 federation 개발과 수행 과정을 절차적으로 제시하고 있으며, 다양한 시간 관리 기법과 HLA test process를 통하여 단순한 일회성 시뮬레이션뿐 아니라 이를 기록, 재생하여 분산 시뮬레이션의 VV&A (Verification, Validation, and Accreditation) 작업이 가능한 점이 있다.

그러나, 디자인 측면 외에 실용성 측면에서 HLA는 아직 많은 점들이 의문인 상태이다. 현재까지의 데모에서는 RTI에 CORBA시스템을 사용하였다. 이런 점들에 의한 RTI에서의 응답 지연이 실시간 시뮬레이션에 적합하지 않으리라는 의구심이 제기되고 있다. 또한 본 연구에서 구현한 모델의 경우, 간단한 생산 시뮬레이션을 모델링한 것임에도 불구하고 일반 컴퓨터에서 수행하기에는 대량의 시스템의 자원을 필요로 하였다. 정상적인 RTI 시스템을 구축하여 사용하려면 전용 서버수준의 시스템을 갖추어 적용함이 바람직하겠다.

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, ALSP등을 대체하겠다는 의도이다. HLA의 연구, 개발에 대한 조기 참여와 지속적 관심이 바람직 할 것이다.

지금까지 미국 국방성에서 시작하여 현재 IEEE의 표준으로 진행되고 있는 HLA/RTI의 생산시스템 시뮬레이션의 적용에 대한 가능성을 알아보았다. 다른 분야에서도 마찬가지이겠지만, 생산시스템에서 발생하는 데이터의 양은 상당히 많이 존재한다. 이와 같은 대규모의 데이터를 관리하기 위해서는 효율성 높은 공통의 네트워크 라이브러리가 필요하다. 또한 실제 생산시스템과 관련된 부분도 만족시킬 수 있어야 한다. RTI는 이러한 요구를 충분히 충족시킬 수 있는 라이브러리이다.

대다수의 첨단 학문의 시작이 그랬듯이 HLA/RTI의 시작 역시 군사적 목적을 위해 처음으로 시도되었다. 따라서, 현재 이 HLA/RTI를 이용하여 개발된 범위도 국방 시뮬레이션을 중심으로 진행되고 있다. 그러나 응용 범위는 대단히 방대하며 모든 분야에 적용이 가능하다. 국방은 물론이거니와 컴퓨터 게임, 엔터테인먼트, 지상 및 항공 교통 등 컴퓨터를 이용한 모든 분산 시뮬레이션과 분산 시스템의 적용이 가능하다.

HLA/RTI의 기본 목적은 분산 시스템간의 상호작용, 즉 분산 시스템간에 서로의 데이터를 주고받는 전송 규약이다. 또 다른 하나의 표준인 셈이다. 서론에서 언급 한 바와 같이 HLA/RTI는 미국 국방성이 중심이 되어 군사 분야의 많은 분야에 적용하여 상당한 효과를 발휘하고 있으며, 현재는 민간 부분에의 적용을 위한 많은 노력을 기울이고 있다. 따라서, HLA/RTI가 컴퓨터를 이용한 상호작용 분야에 주도적인 역할을 할 것으로 예상된다. 이에 반해, 국내의 상황은 아

직 이에 대한 대처가 미약한 상태이다.

현재는 모든 HLA/RTI와 관련된 정보와 application을 무료로 제공하고 있으나, IEEE의 표준으로 확정되고, 사회 전반으로 확산된다면 정보의 획득에 상당한 시간적 노력과 비용이 소비될 것으로 예상된다. 따라서 하루빨리 국내 시장에서도 신속한 대응이 요구된다. 이와 더불어, 우리 자체적으로 국내 상황에 적합한 관련 기술의 개발 및 토착화에 대한 대처도 시급히 요구된다.

참 고 문 헌

[1] 안중호, 「경영과 정보통신 기술」, 학현사, 1994.

[2] Bill Garbacz, Brian Plamondon, Voctor Colon, "Lesson Learned in the SOM and FOM Development Process for a Legacy Simulator", 1999 Spring Simulation Interoperability Workshop, 1999.

[3] Charles Herring, Frederick Hayes-Roth, "The DSSA Process Applied to ADS architecture," Vol. 28, No. 2, 1995.

[4] D. M. Nicol, and Fujimoto. R. M., "Parallel Simulation Today", Annals of Operations Research, Vol. 53, pp. 249-286, 1994.

[5] DMSO, "http://hla.dmso.mil".

[6] DMSO, High Level Architecture Interface Specification.

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

[8] DMSO, High Level Architecture Run-Time Infrastructure Installation Guide RTI 1.3. Version 6., 1999.

[9] Fujimoto, R. M. and D. M. Nicol, "State of the Art in Parallel Simulation", Proceedings of 1992 Winter Simulation Conference, pp. 246-254, 1992.

[10] Fujimoto, R. M., "Parallel Discrete Event Simulation", Communications of the ACM, Vol. 33, No. 10, pp.31-53, 1990.

[11] Jack Ogren, "EAGLE and the High Level Architecture", DMSO, 1997.

[12] Jefferson, D. R., "Virtual Time", ACM Transactions on Programming Language and Systems, Vol. 7, No. 3, pp. 404-425, 1985.

[13] Jim Sikora, Phil Coose, "What in the World is ADS ?", PHALANX, Vol. 28, No. 2, June, 1995.

[14] John A. Hamilton, Jr, et al., Distributes Simulation, CRC Press, 1997.

[15] John W. Shockely, Kirk Parsons, Mark Morgenthaler, "Developing an HLA Virtual Command Post", SIMULATION, Vol. 73, No. 4, November, 1999.

[16] K, Sochats and J. Williams, The Networking and Communications Desk Reference, Presntice-Hall, Carmel, IN, 1992.

[17] Ken Hunt, Dr. Judith Dahmann, Robert Lutz, Jack Sheehan, "Planning for the Evaluation of Automated Tools in HLA", DMSO, 1997.

[18] Lin, Y. and P. A. Fishwick, "Asynchronous Parallel Discrete Event Simulation", IEEE Transactions on System, Man and Cybernetics, Part A, Vol. 26, No. 4, pp. 397-412, 1996.

[19] Marco Cantu, Inside Secretes Delphi 4, SYBEX, 1999.

[20] Mikel D. Petty, Piotr S. Windyga, "A High Level Architecture -based Medical Simulation System", SIMULATION, Vol. 73, No. 4, November, 1999.

[21] Misra, J., "Distributed Discrete-Event Simulation", ACM Computing Surveys, Vol. 18, No. 1, pp. 39-65, 1986.

[22] Pitch Kunskapsutveckling AB, "http://www.pitch.se/"

[23] R. L. Bagrodia, K. M. Chandy, and J. Misra, "A message-based approach to discrete-event simulation," IEEE Transactions on Software Engineering, Vol. 13, No. 6, pp. 664-665, June 1987.

[24] Todd H. Repass, "DIS: An Evolving Modus Operandi for the Department of the Navy," PHALANX, Vol. 28, No. 2. 1995.

[25] Terrain Modeling Project Office,
"http://www.tmpo.nima.mil/guides/Glossary"

[26] The AEgis Technologies Group, Inc. "http://www.aegisrc.com/"

[27] Thomas Schulze, Steffen Stanßburger, Ulrich Klein, "Migration of HLA into Civil Domains: Solution and Prototypes for Transportation Applications", SIMULATION, Vol. 73, No. 4, November, 1999.

부 록

[Federation Execution 생성]

```
rtiAmb.createFederationExecution( fedExecName, "Facility.fed" );
```

[Federation Execution에 연결]

```
federateId = rtiAmb.joinFederationExecution( (char* const) myFacility->GetName(),
                                                    fedExecName, &fedAmb);
```

[turnUpdatesOnForObjectInstance의 결과를 받기 위한 선언]

```
rtiAmb.enableAttributeRelevanceAdvisorySwitch();
```

[초기화 - 실행 중 변경되는 객체의 참조를 위한 핸들을 얻어낸다.]

```
ms_facilityTypeId   = ms_rtiAmb->getObjectClassHandle(ms_facilityTypeStr);
ms_nameTypeId       = ms_rtiAmb->getAttributeHandle( ms_nameTypeStr,
                                                    ms_facilityTypeId);
ms_productTypeId    = ms_rtiAmb->getAttributeHandle( ms_productTypeStr,
                                                    ms_facilityTypeId);
ms_resultTypeId     = ms_rtiAmb->getAttributeHandle( ms_resultTypeStr,
                                                    ms_facilityTypeId);
```

[Publish와 Subscribe]

```
RTI::AttributeHandleSet *facilityAttributes;
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);
facilityAttributes->empty();
delete facilityAttributes;
ms_commTypeId = ms_rtiAmb->getInteractionClassHandle( ms_commTypeStr );
ms_commMsgTypeId = ms_rtiAmb->getParameterHandle( ms_commMsgTypeStr,
                                                  ms_commTypeId );

ms_rtiAmb->subscribeInteractionClass( ms_commTypeId );
ms_rtiAmb->publishInteractionClass( ms_commTypeId );
```

[RTI에 객체를 등록]

```
m_instanceId = ms_rtiAmb->registerObjectInstance( this->GetFacilityRtiId() );
```

[Time Constrained 선언]

```
rtiAmb.enableTimeConstrained();
```

[Time Regulation 선언]

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

[Time Advanced Grant]

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

```
timeAdvGrant = RTI::RTI_FALSE;
rtiAmb.timeAdvanceRequest( requestTime );
```

[변경된 객체를 전송하기 위한 준비와 전송]

```
RTI::AttributeHandleValuePairSet* pNvpSet = this->CreateNVPSet();
(void) ms_rtiAmb->updateAttributeValues( this->GetInstanceId(),
                                         *pNvpSet,this->GetLastTimePlusLookahead(), NULL );

pFacilityAttributes = RTI::AttributeSetFactory::create( 3 );
if ( ( hasNameChanged == RTI::RTI_TRUE ) &&
                                         ( m_sendNameAttrUpdates == RTI::RTI_TRUE ) )
    {
        pFacilityAttributes->add( this->GetNameRtiId(), (char*) this->GetName(),
                                         ((strlen(this->GetName())+1)*sizeof(char)) );
    }

    if ( ( hasProductChanged == RTI::RTI_TRUE ) &&
                                         ( m_sendProductAttrUpdates == RTI::RTI_TRUE ) )
{
    pFacilityAttributes->add( this->GetProductRtiId(), (char*) this->GetProduct(),
                                         ((strlen(this->GetProduct())+1)*sizeof(char)) );
}

if ( ( hasResultChanged == RTI::RTI_TRUE ) &&
                                         ( m_sendResultAttrUpdates == RTI::RTI_TRUE ) )
{
    pFacilityAttributes->add( this->GetResultRtiId(), (char*) this->GetResult(),
                                         ((strlen(this->GetResult())+1)*sizeof(char)) );
}
```

[전송된 자료의 수신과 분리]

```
for ( unsigned int i = 0; i < theAttributes.size(); i++ )
    {
        attrHandle = theAttributes.getHandle( i );
    if ( attrHandle == Facility::GetResultRtiId() )
    {
        char result[ 1024 ];
        theAttributes.getValue( i, (char*)result, valueLength );
        SetResult( (const char*)result );
    }
    else if ( attrHandle == Facility::GetProductRtiId() )
    {
        char product[ 1024 ];
        theAttributes.getValue( i, (char*)product, valueLength );
        SetProduct( (const char*)product );
    }
    else if ( attrHandle == Facility::GetNameRtiId() )
    {
        char name[ 1024 ];
        theAttributes.getValue( i, (char*)name, valueLength );
        SetName( (const char*)name );
    }
}
```

[Federation Execution에서 객체 삭제]

```
rtiAmb.resignFederationExecution(
                                RTI::DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES );
```

[Federation Execution의 소멸]

rtiAmb.destroyFederationExecution(fedExecName);

ABSTRACT

An Application of HLA/RTI in Manufacturing Simulation

Kwon, Soon Jong

Department of Industrial Engineering

The Graduate School of

Hansung University

The purpose of this paper is to study the applicability of the concept and methodology of HLA/RTI to the industrial manufacturing system simulation. HLA/RTI is a new system design methodology for distributed simulation and it is only a short time since the people recognized powerful in this community.

HLA, a common library for network simulation, was developed for activity and effectiveness among the simulation modules by DMSO. 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. RTI can be enough satisfying library and tools.

As a result of study in this thesis, a possibility of applying of the HLA methodology to a small manufacturing system turned out to be acceptable and may be available in some other distributed systems.

Especially, in data exchange through network HLA is the rule to core part.

An enlargement of HLA technique may contribute to other systems that have something to do with networking, for example, on-line game, traffic control problems, industrial logistics, communication, etc. Just now, HLA/RTI is understanding as a new solution to the network problems.

감사의 글

그 동안 힘들게 얻은 지식을 가지고 다시 또 새로운 세상을 해쳐나갈 수 있는 힘을 주신 분들께 짧지만 지면을 빌어 감사의 마음을 전하고자 합니다.

새로운 세계를 접하는 두려움과 기대 속에서 정도를 벗어나지 않도록 도와주시고, 새로운 학문의 길로 인도해 주신 홍윤기 교수님께 깊은 감사를 드립니다. 또한 대학원 생활의 처음부터 논문의 마지막까지 충고와 관심을 기울여주신 김대홍 교수님, 정병용 교수님, 박명환 교수님, 위남숙 교수님, 원형규 교수님, 이재득 교수님, 유재건 교수님 그리고 홍정완 교수님께도 감사드립니다.

혼자일 것 같았던 대학원 생활을 같이 생활해 주고 힘들 때마다 조언을 아끼지 않은 대학원 동기 모창우, 신연봉에게도 고맙다는 말을 전하고 싶습니다. 그리고 뒤늦게나마 들어온 나이 많은 후배인 유운식과 비록 자주 만나지는 못했지만 이정주에게도 고마움을 표합니다.

부족한 선배를 옆에서 보면서 많은 도움을 주었던 신찬수, 김재경, 홍상용, 송준상, 곽대월을 비롯한 많은 후배들에게 정말로 감사드립니다.

바쁘다는 것을 핑계삼아 만나지도 못했던 저에게 끝까지 용기를 북돋아 준 친구들, 특히 한동석군을 비롯한 예전 ACA 회원들에게 미안하다는 말과 고맙다는 말을 함께 전합니다.

마지막으로 항상 밤늦게 집에 돌아가도 따뜻한 말로 격려를 해 주신 부모님, 사랑합니다.

1999년 12월

권 순 종