# 공유메모리를 사용한 레거시 원자력 시뮬레이션 코드의 HLA 패더레이션으로의 통합

박 근 옥[†] · 한 관 호[††] · 임 종 태[†††]


## 요 약

미국 국방성에서 주관한 시뮬레이션 표준인 HLA(High Level Architecture)의 목적은 시뮬레이션 소프트웨어들 사이의 상호 호환을 용이하게 하고 그들 구성 요소들의 재사용을 촉진하는데 있다. 산업 현장에는 HLA가 시뮬레이션 표준이 되기 이전에 개발된 많은 시뮬레이션 소프트웨어들이 있다. 레거시 시뮬레이션들을 HLA를 사용한 패더레이션으로의 통합은 M&S(Modeling & Simulation) 영역에서 중요한 연구 주제이다. 원자력과 우주항공 같은 임무 완수가 중요한 산업의 레거시 시뮬레이션 소프트웨어들은 일반적으로 Fortran 언어를 사용한다. 하지만 HLA가 Fortran 언어를 지원하고 있지 않기 때문에 그들의 재사용은 쉽지 않다. 본 논문은 레거시 시뮬레이션 소프트웨어의 변경을 최소화하면서 HLA 패더레이션으로 이전을 용이하게 하는 통합 방법을 제시한다. 패더레이션에 참여하는 각 패더레이트는 실행 시간에 생성되는 공유메모리를 통하여 통신하는 분리된 실행을 갖는다. 발행과 접수를 위한 두 가지 유형의 공유메모리 블록이 사용된다. 레거시 시뮬레이션 소프트웨어에서 사용되는 전역변수 선언 블록은 발행과 접수를 위하여 분할되고 HLA FOM 설계를 위하여 객체 및 상호작용 클래스로 사상된다. 제안된 방법을 검증하기 위하여 플랜트 설계에 사용되고 있는 레거시 원자력 시뮬레이션 코드의 HLA 통합을 시도하였고 통합 결과를 관측하기 위하여 FMT(Federation Management Tool)를 사용하였다. FMT가 표시하는 진단정보는 본 연구가 제안하는 방법이 성공적이고 효과적으로 HLA 통합에 사용될 수 있음을 보였다.


**키워드 : 공유메모리, 패더레이션, 패더레이트, 원자력 시뮬레이션, HLA, FOM**

# An Integration of Legacy Nuclear Simulation Code into HLA Federation using Shared Memory

Geun-Ok Park[†] · Kwan-Ho Han[††] · Jong-Tae Lim[†††]


## ABSTRACT

The objective of the HLA(High Level Architecture) have recommended by DoD(Department of Defense) is to facilitate interoperability among simulations and to promote reuse of their components. There are many legacy simulation softwares developed before the HLA becomes simulation standard. The integration of legacy simulations into federations using the HLA is an important research topic in M&S(Modeling and Simulation) area. Legacy simulation softwares of the mission critical industry such as nuclear and aerospace are generally use Fortran language. However, the reuse of those is not easy because the HLA is not support Fortran language. This paper suggests a integration method which minimizes the modification of legacy simulation software and migrates the legacy simulation software to HLA federation. Each federate participating in federation have the separated executables that communicate via a shared memory created at run-time. Two types of shared memory blocks are used for publication and subscription. Declaration block for global variables used in legacy simulation software is separated for publication and subscription and then mapped as classes of objects and interactions for the HLA FOM design. To validate the suggested method, we approached the HLA integration of legacy nuclear simulation code being used in plant design and to observe the integration results, we used the FMT(Federation Management Tool). The diagnostic information which the FTM displays showed that our method can be successfully and effectively used for a HLA federation.


**Key Words : Shared Memory, Federation, Federate, Nuclear Simulation, HLA, FOM**

## 1. 서 론

시뮬레이션 표준은 SIMNET(Simulator Networking; SIMNET), ALSP(Aggregate Level Simulation Protocol; ALSP), DIS(Distributed Interactive Simulation : DIS), HLA

† 정 회 원 : 공주대학교 컴퓨터공학과
†† 준 회 원 : 공주대학교 컴퓨터공학과
††† 종신회원 : 공주대학교 컴퓨터공학부 교수
논문접수 : 2005년 5월 12일, 심사완료 : 2005년 8월 3일

(High Level Architecture; HLA) 순서로 발전되어 왔다. 발전과 변천의 배후에는 국방산업 주도 영역의 시뮬레이션 소프트웨어(또는 코드)에 대한 투자비용 절감, 재사용성, 확장성, 상호호환성 등의 요구를 충족시키려는 시대적 요구가 작용하였다[1]. 미국 국방성에 의하여 M&S(Model and Simulation) 표준으로 지정된 HLA는 기존 시뮬레이션 표준의 패러다임에 대한 중대한 변화로 인식되고 있으며 국방산업을 벗어나 일반산업에도 적용할 수 있도록 2000년에 IEEE 1516 표준으로 제정되었다[2].

웹 기반 관련 기술의 발전과 HLA의 확산에 따라 상호 영역간의 기술을 융합함으로써 보다 재사용성 있고 상호 호환성 있는 웹 기반 M&S를 체계를 구축하기 위한 노력의 일환으로 XMSF(Extensible Modeling and Simulation Framework; XMSF)가 제안되었다[3]. XMSF에서 토의되고 제안된 시급히 해결해야 할 기술적 현안 중의 하나는 레거시(legacy) 시뮬레이션의 HLA 패더레이션(federation)으로의 이전 및 통합을 통한 재사용성의 촉진이다.

패더레이션이란 HLA 표준 규격에 부합되게 개발되고 운영될 수 있는 모의된 체계이며 다수의 패더레이트(federate)들로 구성된다. 패더레이트는 패더레이션에 참여하는 개별적인 시뮬레이션(또는 모의된 객체) 소프트웨어이다. 레거시 시뮬레이션이 HLA 패더레이션에 참여하려면 HLA 표준규격에서 기술된 요건을 충족해야 한다. HLA 표준 규격은 규칙(rule), 인터페이스 사양(interface specification), OMT(Object Model Template)의 세 부분으로 구성되어 있다.

HLA 규칙은 패더레이션 실행 시에 각 패더레이트 간에 일어나는 상호작용을 달성하기 위한 규약을 기술하고 있으며, 패더레이트를 위한 5개의 규칙과 패더레이션에 적용되는 5개의 규칙으로 구성되어 있다[4]. 인터페이스 사양은 패더레이트 간의 상호작용을 위한 서비스로써 패더레이션 관리, 선언(declaration) 관리, 객체 관리, 소유권(ownership) 관리, 데이터 분배(distribution) 관리, 시간 관리로 구성된 6개 관리 그룹으로 구성되어 있다[5]. 미국 국방성은 개발자의 편의를 위하여 인터페이스 사양을 구현한 RTI(Runtime Infrastructure; RTI)와 API(Application Program Interface; API)를 제공하고 있다. 현재의 인터페이스 사양과 API는 C++, Ada, Java, 그리고 CORBA IDL의 언어에 대해서만 정의되어 있다. OMT는 패더레이션의 객체 클래스 구조 및 상호작용, 속성, 파라미터, 자료사전 등을 표현하기 위한 표준화된 자료 형식을 규정하고 있는 프레임워크이다. HLA OMT는 SOM(Simulation Object Model; SOM), FOM(Federation Object Model; FOM), MOM(Management Object Model : MOM)의 세 가지 모델이 있다. 각 모델은 6개의 테이블로 구성되어 있다[6].

HLA 이전의 시뮬레이션 표준에 따라 개발된 레거시 시뮬레이션 소프트웨어들은 HLA 표준 규격과 너무나 상이하여 레거시 시뮬레이션의 HLA 패더레이션으로의 통합은 쉽지 않다. 인터페이스 사양이 C++, Ada, Java, 그리고 CORBA IDL 언어만을 규정하고 있으므로 Fortran 언어를 많이 사용하는 항공우주, 원자력, 실시간 산업 플랜트 영역의 시뮬레이션 소프트웨어의 재사용에 대한 어려움을 가중시킨다. 특히 HLA는 다중방송(multi casting) 통신을 사용하고 DIS 등의 이전 시뮬레이션 표준은 광역 방송(broad casting) 통신을 사용하므로 RTI 서비스와 API를 적절히 이용한 레거시 시뮬레이션의 기술적 통합 노력이 요구된다.

본 연구는 레거시 시뮬레이션 소프트웨어의 변경을 최소화하면서 HLA 패더레이션으로 이전을 용이하게 하는 통합 방법을 제시한다. 본 논문의 제 2 장에서는 레거시 시뮬레이션의 HLA 패더레이션으로의 통합 전략 유형과 관련 연구 사례를 살펴보고, 3장에서는 패더레이트들이 상호 관심을 갖는 데이터만을 발행(publication)과 접수(subscription) 개념에 근거하여 분할한 공유메모리를 통해 교환함으로써 레거시 시뮬레이션이 HLA 패더레이션으로 용이하게 통합 가능한 구조를 제시한다. 4장에서는 본 연구에서 제시한 구조가 현실세계에서 타당성 있게 활용 가능함을 보이기 위하여 실시간 속성을 갖는 원자력 시뮬레이션 코드를 대상으로 설계 및 구현한 내용을 기술한다. 5장에서는 원자력 레거시 시뮬레이션 코드를 HLA 패더레이션으로 통합하여 실행시킨 결과를 논의하고 6장에서는 결론을 맺는다.

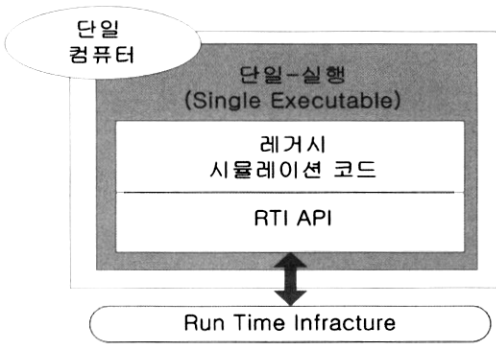
## 2. 관련 연구

### 2.1 레거시 시뮬레이션의 통합 전략 유형

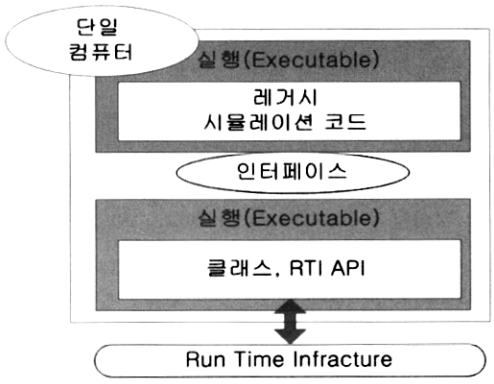
패더레이트 그 자체는 패더레이트 코드와 LRC(Local RTI Components)로 구성된다[7]. RTI 컴포넌트는 HLA 인터페이스 사양을 특정한 프로그래밍 언어로 구현한 것이다. 패더레이트 코드는 개발자가 작성한 시뮬레이션 코드와 FederateAmbassador 클래스로 구성된다. 실세계의 물리적, 열수력학적, 동적 특성과 현상을 특정한 프로그램 언어로 작성하여 구현한 시뮬레이션 코드는 HLA 인터페이스 사양을 따르는 FederateAmbassador 클래스와 RTI API를 이용하여 LRC에게 서비스를 요청한다.

LRC는 HLA 인터페이스 사양을 프로그래밍 언어로 구현한 것으로 RTIAmbassador 클래스를 포함하며, RTI 인터페이스 사양에서 정의하고 있는 6개 관리 그룹에 대한 서비스를 제공한다. 패더레이트 코드의 서비스 요청은 RTIAmbassador 클래스에 의하여 처리되고 처리결과는 FederateAmbassador 클래스에게 반환된다.

레거시 시뮬레이션의 재사용 관점에서 시뮬레이션 코드가 반드시 FederatorAmbassador 및 RTIAmbassador의 클래스와 RTI 서비스를 사용해야 한다는 점에서 어려움이 발생한다. 특히 재사용하려는 시뮬레이션 코드가 C++, Ada, Java, CORBA IDL로 작성되지 않은 경우에는 RTI 서비스를 사용하기 위한 특별한 통합 전략이 고안되어야 한다. 레거시 시뮬레이션의 HLA 패더레이션 통합은 단일 컴퓨터-단일 실행(single executable), 단일 컴퓨터-복수 실행(multiple executables), 복수 컴퓨터-복수 실행의 3 가지 유형으로 구분할 수 있다[8].




(그림 1) 단일 컴퓨터-단일 실행 통합

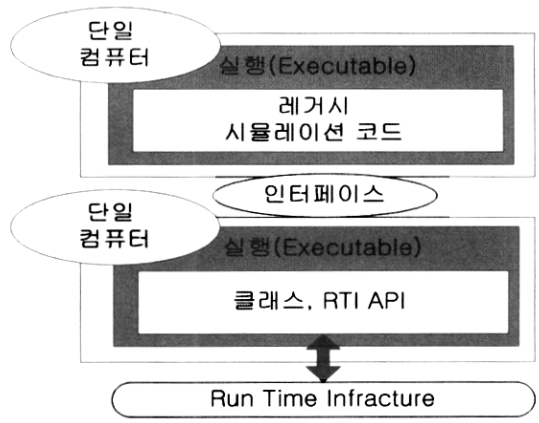



(그림 2) 단일 컴퓨터-복수 실행 통합

(그림 1)의 단일 컴퓨터-단일 실행 형태는 레거시 시뮬레이션 코드와 RTI API를 긴밀하게 결합시켜 단일의 컴퓨터 상에서 실행이 가능한 하나의 실행 파일 형태로 패더레이트를 생성한다. 이 방법은 시뮬레이션 실행 속도 측면에서 우수하다는 장점을 갖는다. 반면에 모든 원시 코드는 RTI API를 직접 이용할 수 있도록 C++, Ada, Java, CORBA IDL 중 하나의 언어를 사용하여 작성되어야 한다. Fortran과 같이 RTI API를 직접 이용할 수 없는 레거시 시뮬레이션 코드는 RTI API를 이용할 수 있는 언어로의 변환이 필요하다.

(그림 2)의 단일 컴퓨터-복수 실행 형태는 레거시 시뮬레이션 코드에 대한 변경 및 수정을 적게 하면서 RTI 서비스를 이용할 수 있도록 레거시 시뮬레이션 코드와 RTI API 사이에 인터페이스를 두는 방법이다. 이 방법은 레거시 시뮬레이션 코드가 인터페이스에 접근할 수 있게 하는 기능 호출(function call) 루틴(routine)을 작성하여 시뮬레이션 원시 코드의 일부가 되도록 삽입한 후 기능 호출의 삽입에 따른 필요한 라이브러리의 컴파일과 링크, 인터페이스를 경유한 RTI 서비스 사용을 위한 호출 등의 추가적인 원시 코드의 작성이 필요하다. 최근의 연구사례들은 인터페이스 구현을 위한 수단으로 공유메모리를 많이 사용한다.

(그림 3)의 복수 컴퓨터-복수 실행 통합 전략은 레거시 시뮬레이션 소프트웨어가 실행되는 컴퓨터상에 RTI 설치가 불가능한 경우에 적용하는 방법이다. 이 방법에서 인터페이스는 게이트웨이(gateway)의 역할을 수행한다. 게이트웨이 기능은 레거시 시뮬레이션이 실행되는 컴퓨터 또는 RTI 서비스가 제공되는 컴퓨터의 어느 컴퓨터에서도 구현될 수 있다. 이 통합 전략은 시뮬레이션 실행 속도가 보장되지 않는다는 단점이 있다. 또한 게이트웨이의 기능이 개발자의 재량에 따라 구현되므로 HLA가 의도하는 재사용성이 저해된다.




(그림 3) 복수 컴퓨터-복수 실행 통합

### 2.2 HLA로의 통합 연구사례

(그림 1)의 통합 전략을 적용하여 Fortran 77 언어로 작성된 레거시 시뮬레이션 코드에 F2J(Fortran to Java)라는 자동변환 도구를 적용하여 Java 언어로 구성되는 패더레이트를 생성하려는 시도는 시간이 과도하게 소요되고 결과적으로 얻은 Java 컴포넌트는 본래의 레거시 시뮬레이션 코드보다 느리게 실행되는 문제점을 보였다[9]. Fortran 77 소스 코드가 Java 코드로 자동변환 되었으나 Fortran 77에서 사용되는 모든 데이터 형과 제어 흐름이 올바르게 동작하는지를 확신할 수 없었을 뿐만 아니라 변환된 시뮬레이션 코드가 정확하게 동작함을 입증하기 위하여 레거시 시뮬레이션 코드 개발 시에 수행하였던 것과 같은 광범위한 재시험 수행이 요구되었다[9].

국립우주생의학연구소(National Space Biomedical Research Institute)가 심장 혈류의 상태 실험 및 관측을 위하여 연구 개발한 CVVS(CardioVascular Ventricular System) 패더레이션은 C++ 코드와 Fortran 코드를 호출하여 결합하는 방법을 사용하여 단일 실행 가능한 패더레이트를 생성하였다[10]. (그림 1)의 통합 유형으로 분류할 수 있는 이 연구는 C++ 언어를 사용한 Fortran 95 서브루틴 호출 코드 작성, Fortran 95 서브루틴 컴파일, C++ 코드와 컴파일된 Fortran 95의 링킹 및 실행 파일 생성으로 구성된 일련의 작업을 수행하였다. 작업 과정에서 Fortran 컴파일러의 특성에 부합시키기 위한 Fortran 소스 코드의 수정, 서로 다른 언어 사용에 따른 서브루틴 호출 시의 매개변수 전달 차이점 조정을 위한 부가적인 노력이 있었다.

RCVSIM(Revised CVSIM)은 Mathworks 사의 MATLAB을 이용하여 MIT가 개발한 CVSIM(CardioVascular SIMulator)

을 HLA 패더레이션으로 통합한 사례로 (그림 2)의 통합 전략을 사용하였다. MATLAB으로 작성되고 컴파일된 시뮬레이션과 HLA 인터페이스가 각각 분리된 상태로 단일 컴퓨터에서 실행되며 양자는 공유메모리를 사용하여 상호 통신한다[11]. 이 연구는 공유메모리에 대한 판독과 기록의 통제를 위하여 플래그(flag)를 사용한다. 서로 분리된 실행이 공유메모리의 접근 권한을 얻기 위하여 플래그의 설정과 해제를 위해 경쟁하고 불특정한 시간동안 대기해야 한다. 마감시한(deadline) 제약을 갖는 실시간 시뮬레이션의 경우 예측할 수 없는 대기시간은 허용되지 않으므로 플래그를 사용한 공유메모리 통신은 바람직하지 않다.

(그림 3)의 통합 전략을 적용한 레거시 시뮬레이션 코드 재활용 사례가 있다[12]. 이 연구는 시뮬레이션 실행 속도가 보장되지 않는 단점을 보상하기 위하여 게이트웨이에 XDR (External Data Representation) 필터를 사용한 특화된 C++ 라이브러리를 개발하고 별도의 컴퓨터에 게이트웨이 기능을 구현하였다. 신속한 통합과 저 비용을 고려하여 게이트웨이 방법을 사용한 연구이나 XDR은 HLA 표준사양이 아닐 뿐만 아니라 자신들만의 특화된 라이브러리를 사용함으로써 재사용 문제를 여전히 내포하고 있다.

국내의 경우 HLA 연구는 국방 분야를 중심으로 최근 수행되고 있으며 일반 산업계는 태동기라 할 수 있다[13]. 창조21이라는 패더레이션은 미국 국방성의 인증을 받은 국내 최초의 사례이다. 창조21은 자신들의 응용에 특화된 미들웨어 수준의 변환기를 개발하여 (그림 1)의 유형에 해당하는 통합과 병행하여 일부의 레거시 시뮬레이션에는 (그림 3)의 통합 방법으로 HLA 패더레이션을 개발하였다. 또 다른 국내 연구로 소켓 인터페이스를 사용하여 이미 개발된 내장형 시스템을 HLA 패더레이션에 연동시킨 사례가 있다[14]. 이 연구는 (그림 3)의 통합 유형에 속한다.

레거시 시뮬레이션을 패더레이션에 통합한 기존의 연구사례들은 결과적으로 통합을 달성하였으나 HLA가 추구하는 기술적 특성인 발행과 접수의 개념을 활용하지 못하고 있다. 본 연구는 (그림 2)의 통합 전략을 채택하고 공유메모리에 발생과 접수 개념을 적용한 연구를 수행한다.

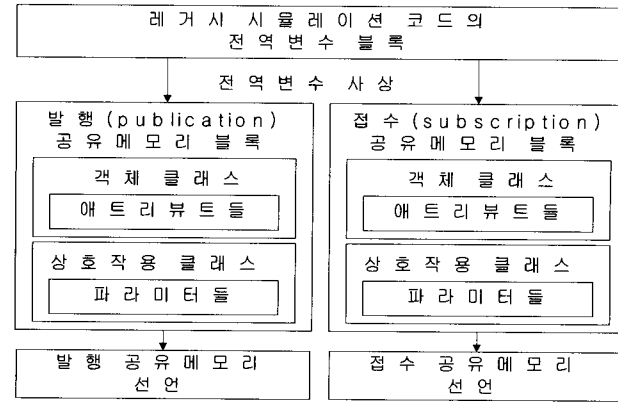
## 3. 공유메모리 기반 패더레이트 구조

### 3.1 전역변수의 공유메모리 사상

공유메모리는 낮은 지연, 결정론적, 높은 대역폭 내부 프로세스 통신을 제공하므로 마감시한 제약을 갖는 실시간 시뮬레이션 영역의 응용에 많이 사용된다. 원자력 시뮬레이션 코드의 경우에도 공유메모리를 사용하여 프로세스 간 통신을 수행한다[15, 16]. 우리의 연구팀은 본 연구의 사전연구로써 원자력 시뮬레이션 코드의 개발 요건과 공유메모리를 사용한 응용 소프트웨어의 구조를 제시하였다[17].

HLA 표준규격은 패더레이트들 간에 서로 관심을 갖는 데이터만 교환되도록 RTI 서비스를 정의하였다. 또한 데이터의 값이 갱신되는 경우에만 관심을 갖는 패더레이트에게 데이터를 전송한다. HLA는 교환과 전송에 필요한 데이터를 위하여 객체 클래스(object class)와 상호작용 클래스(interaction class)를 사용한다. 객체 클래스는 애트리뷰트의 집합이며, 상호작용 클래스는 파라미터의 집합이다[2]. 객체 클래스는 시뮬레이션이 진행되는 동안에 지속되는 데이터 값을 갖는 애트리뷰트(attribute)들로 구성된다. 반면에 상호작용 클래스는 특정 시뮬레이션 사건이 발현되는 순간에만 의미있는 값을 갖고 사건이 종료되면 소멸되는 데이터 값을 갖는 파라미터(parameter)들로 구성된다.

레거시 원자력 시뮬레이션 코드의 경우 C++와 Fortran 언어가 사용되고 동일 컴퓨터나 다른 컴퓨터에서 실행되는 프로세스들 간에 교환되는 데이터는 전역변수(global variable)로 선언되어 사용된다. 따라서 본 연구는 전역변수의 속성을 파악하여 객체 클래스와 상호작용 클래스로 사상(mapping)시키고 사상시킨 전역변수를 발행(publication)과 접수(subscription) 개념을 갖는 공유메모리에 할당하여 상호간에 통신하는 패더레이트 구조를 제안한다. 전역변수의 객체 및 상호작용 클래스로의 사상 과정은 (그림 4)와 같다.




(그림 4) 전역변수의 공유메모리 사상 개념

전역변수를 공유메모리에 사상시킬 때 HLA RTI가 제공하는 발행과 접수 서비스를 효율적으로 사용할 수 있도록 발행 공유메모리 블록과 접수 공유메모리 블록으로 양분시킨다. 발행 공유메모리 블록은 관심을 갖는 다른 페더레이트에게 레거시 시뮬레이션 코드가 제공해야할 전역변수들로 구성되며, 레거시 시뮬레이션 코드만이 전역변수에 대한 갱신권한을 갖는다. 접수 공유메모리 블록은 레거시 시뮬레이션 코드가 다른 페더레이트로부터 받아야할 전역변수들로 구성되며, 레거시 시뮬레이션 코드는 이 전역변수의 값에 대한 판독권한만 갖는다. 이와 같이 공유메모리 블록을 양분시킴으로써 단일 컴퓨터상에서 실행될 레거시 시뮬레이션 코드와 RTI 서비스 간의 공유메모리 점유 경쟁에 따른 대기시간을 회피할 수 있다. 각각의 공유메모리 블록은 패더레이트 구성요소로서의 역할을 수행하기 위하여 객체 클래스와 상호작용 클래스로 구성한다. 각 클래스에는 <표 1>의 기준에 따라 해당되는 전역변수가 애트리뷰트와 파라미터로 할당된다.

```
extern "C" struct {
        struct  {
                int  ismode_req;
                int  malfunction_req;
                  - - -
                double sim_time;
               ] double corepower[10];
                  - - -
               } EX;
             } EXAMP;
```

(그림 5) C++ 언어의 공유메모리 블록 선언

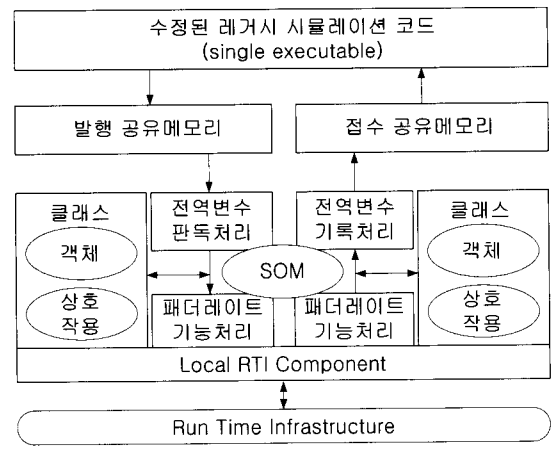
```
MODULE EXAMP
  TYPE MYDATA
    SEQUENCE
      integer(kind=4) &
          ismode_req, malfunction_req
                - - -
      real(kind=8) &
          sim_time, corepower[10]
                - - -
    END TYPE MYDATA
  TYPE (MYDATA) EX
END MODULE EXAMP
```

(그림 6) Fortran 언어의 공유메모리 블록 선언

사상이 완료된 발행과 접수 블록은 레거시 시뮬레이션 코드가 공유메모리를 사용하여 RTI 서비스와 통신할 수 있도록 (그림 5) 및 (그림 6)과 같은 형태로 특정 프로그래밍 언어를 사용하여 선언된다. (그림 5)는 패더레이트의 구성요소인 클래스 및 RTI 서비스가 C++ 언어를 사용하여 개발되는 경우의 블록을 구조체 형태로 선언한 일례이다. (그림 6)은 Fortran 언어를 사용하는 레거시 원자력 시뮬레이션 코드의 전역변수들을 모듈 형태로 선언한 일례이다. (그림 5)의 구조체 내부를 구성하는 전역변수와 (그림 6)의 모듈 내부를 구성하는 전역변수는 일대일 대응관계를 유지한다. Fortran 언어로 작성된 원시코드의 전역변수 영역만을 대상으로 (그림 6)과 같이 몇 줄의 코딩을 추가한 수정된 원시코드를 Fortran 컴파일러로 컴파일함으로써 공유메모리 통신이 가능한 단일 실행 레거시 시뮬레이션 코드를 얻을 수 있다.

### 3.2 레거시 시뮬레이션의 패더레이트 구조

레거시 시뮬레이션에 참여하는 개개의 패더레이트는 (그림 7)과 같이 수정된 레거시 시뮬레이션 코드, 공유메모리, 클래스, SOM, 전역변수 판독 및 기록처리, 패더레이트 기능처리, LRC로 구성된다. 수정된 레거시 시뮬레이션 코드는 단일 실행 가능한 형태이며, 이를 제외한 나머지 부분인 패더레이트 코드도 하나의 단일 실행 기능한 형태이다. 따라서 (그림 2)의 통합 유형과 같이 한 대의 컴퓨터에 두 개의 단일 실행이 존재하며 양자는 공유메모리 블록을 통하여 상호 내부통신을 수행한다. 패더레이트 코드는 RTI 인터페이스 사양에서 규정하고 있는 서비스를 구현한 LRC를 사용하여 RTI에 접속하고 다른 패더레이트와 상호 관심 있는 데이터를 교환한다.




(그림 7) 패더레이트의 구성요소

〈표 1〉 전역변수의 객체 및 상호작용 클래스 할당 기준

| 블록 구분 | 클래스 구분 | 전역변수의 속성/용도 |
|---|---|---|
| 발행 블록 | 객체 (애트리뷰트) | 레거시 시뮬레이션 코드가 지속적이고 주기적으로 갱신하는 전역변수 |
| | 상호작용 (파라미터) | 레거시 시뮬레이션 코드가 이산적이고 산발적으로 갱신하는 전역변수 |
| 접수 블록 | 객체 (애트리뷰트) | 레거시 시뮬레이션 코드가 다른 코드로부터 지속적이고 주기적인 연산처리에 사용할 것을 요구받는 전역변수 |
| | 상호작용 (파라미터) | 레거시 시뮬레이션 코드가 다른 코드로부터 일회성 갱신을 요구받는 변수 |

클래스 내부는 <표 1>의 기준에 따라 사상되고 분류된 객체 애트리뷰트와 상호작용 파라미터로 구성되며, HLA 규칙과 MOM의 사양에 따라 패더레이트 당 하나의 SOM으로 표현된다. 패더레이트 기능처리 부분은 공유메모리로부터 판독하거나 공유메모리에 기록해야할 전역변수가 관심을 갖는 패더레이트들 간에 발행 및 접수될 수 있도록 Federate Ambassador 클래스와 RTI API로 작성한 원시코드로 구성된다. 또한 기능처리 부분은 패더레이트 자신의 지역변수 값과 공유메모리 변수 값을 사용해 추가로 처리해야할 특화된 처리(예 : 그래픽 화면의 시각적 표시, 데이터 수집 등)를 위한 원시 코드도 포함한다.

## 4. 원자력 시뮬레이션 코드의 패더레이션 통합

### 4.1 레거시 시뮬레이션 코드의 현황

본 연구가 제안하는 패더레이트 구조를 적용하여 HLA 패더레이션으로 통합하려는 원자력 시뮬레이션 코드는 현재 발전소 설계 및 건설에 활용 중인 실시간 시뮬레이션 코드이다[15, 16]. 이 코드는 약 300개의 Fortran 서브루틴으로 구성되어 있으며 원자로의 동적 반응특성, 발전소의 열수력 특성을 시뮬레이션한다[18].

본 연구에서 사용하는 원자력 시뮬레이션 코드는 통신망에 접속된 각 컴퓨터가 자신에게 부여된 시뮬레이션 기능처

〈표 2〉 광역 방송 전역변수의 크기와 개수

| 코드구성 모듈이름 | 광역 방송 데이터 | |
|---|---|---|
| | 전역변수 개수 | 크기(byte) |
| 동특성 엔진 모듈 | 569 | 4552 |
| 강사 모듈 | 14 | 112 |
| 운전원 모듈 | 84 | 672 |
| (합계) | 667 | 5,336 |

리를 수행하고 코드를 구성하는 모듈들 간에 요구되는 모든 데이터(전역변수들)의 송신 및 수신을 1 초 이내에 완료해야 하는 실행 시간제약을 갖는다. 만약에 시간제약을 만족하지 못하면 오류 메시지가 발행되고 시뮬레이션은 중지된다. 현재 이 코드는 전형적인 레거시 시뮬레이션처럼 광역 방송 통신 방식인 DIS 형식으로 컴퓨터에서 실행되는 각 모듈이 갖는 모든 전역변수 값을 무조건 통신망으로 출력하기 때문에 통신 부하가 크다. 또한 시뮬레이션 기능 확장을 위하여 새로운 전역변수를 모듈에 추가하거나 새로운 컴퓨터를 통신망에 추가할 때 통신 구조를 세심하게 재설계해야 하는 문제를 내포하고 있다. <표 2>는 현재의 레거시 시뮬레이션 코드를 구성하는 각 모듈들이 1초 마다 통신망으로 데이터를 광역 방송하는 현황이다. 개개의 전역변수는 각각 8 바이트 크기를 갖는다.

### 4.2 전역변수의 공유메모리 사상과 설계

<표 3>은 레거시 원자력 시뮬레이션 코드에서 사용하는 전역변수를 <표 1>의 기준에 따라 객체와 상호작용 클래스로 사상시키고 발행과 접수의 공유메모리 블록으로 할당한 설계 결과이다. 동특성 엔진은 시뮬레이션이 진행되는 동안에 온도, 유량, 압력, 속도, 제어밸브의 열림 정도 등과 같은 물리적 속성을 표현하는 전역변수의 값을 계속 변경시키므로 이의 속성에 대응하는 전역변수는 객체 클래스의 애트리뷰트로 사상시킬 수 있다. 반면에 경보, 격리밸브의 닫힘과 열림 같은 물리적 속성은 값의 변화 시점을 예측할 수 없고 변화 또한 산발적이므로 이에 대응하는 전역변수는 상호작용 클래스의 파라미터로 대응시킬 수 있다. 동특성 엔진에 의하여 변화되는 전역변수 값의 변화는 운전원 및 강사가 감시 및 진단할 수 있도록 운전원(Operator) 및 강사(Instructor) 모듈에 전송되어야하므로 발행 공유메모리 블록으로 할당한다.

강사(Instructor) 모듈은 동특성 엔진과 운전원 모듈의 시뮬레이션 실행을 통제하는 명령을 생성하고 동특성 엔진과 운전원 모듈로부터 해당 명령에 대한 응답의 상태를 판단한다. 강사가 사용하는 명령은 강사의 판단에 의하여 수시로 발생하는 속성을 갖는다. 따라서 이에 해당하는 전역변수는 강사 모듈이 시뮬레이션 엔진과 운전원 모듈을 대상으로 발행하는 상호작용 파라미터로 정의할 수 있다. 또한 강사 명령에 대한 응답 관련 전역변수는 동특성 엔진과 운전원 모듈이 강사 모듈을 대상으로 발행하고 강사 모듈이 접수해야 하는 상호작용 파라미터로 정의할 수 있다. 시뮬레이션 엔

〈표 3〉 전역변수의 클래스 사상과 할당

| 전역변수의 속성/용도 | 발행자 | 접수자 | 객체/상호작용 | 애트리뷰트/파라미터 (개수) |
|---|---|---|---|---|
| 동특성 엔진의 온도, 유량, 압력, 속도, 제어밸브의 열림 정도, 연료의 연소도 등 | 동특성 엔진 | 운전원 | 객체 | 애트리뷰트 (460) |
| 동특성 엔진의 경보, 격리밸브의 닫힘과 열림 등 플랜트의 주요 운전사건 변화 등 | 동특성 엔진 | 운전원, 강사 | 상호작용 | 파라미터 (97) |
| 강사 모듈이 시뮬레이션 엔진과 운전원 모듈의 실행 제어를 통제하는 명령 등 | 강사 | 동특성 엔진, 운전원 | 상호작용 | 파라미터 (12) |
| 강사 명령에 대하여 시뮬레이션 엔진과 운전원 모듈이 반응하는 응답 등 | 동특성 엔진, 운전원 | 강사 | 상호작용 | 파라미터 (12) |
| 시뮬레이션 초기화, 개시, 일시중지, 동기화의 시간관리 명령 | 강사 | 동특성 엔진, 운전원 | 객체 | 애트리뷰트 (2) |
| 시간관리 명령에 대한 시뮬레이션 엔진과 운전원 모듈의 응답 | 동특성 엔진, 운전원 | 강사 | 객체 | 애트리뷰트 (2) |
| 운전원이 발전소를 제어하는 운전 관련 행위 관련 명령 | 운전원 | 동특성 엔진 | 상호작용 | 파라미터 (70) |

진과 운전원 모듈의 실행과 관계된 시간관리 전역변수는 주기적인 갱신과 지속적 속성을 가지므로 강사 모듈이 발행하고 시뮬레이션 엔진과 운전원 모듈이 접수해야하는 객체 애트리뷰트로 할당함이 타당하다.

운전원 모듈은 동특성 엔진이 처리하여 발행한 전역변수의 값을 운전원이 판독하고, 이를 바탕으로 운전원이 결정한 적절한 제어행위 처리 요구를 동특성 엔진에게 전달하는 기능을 수행한다. 따라서 이와 관련된 전역변수는 운전원 모듈이 발행하고 동특성 엔진이 접수해야하는 상호작용 파라미터로 사상시킬 수 있다.

발행과 접수를 위한 사상이 완료되면 (그림 5)와 (그림 6)의 방식으로 레거시 시뮬레이션 원시 코드에 C++ 언어와 Fortran 언어를 사용하여 발행과 접수 공유메모리 블록으로 전역변수를 선언한다. 또한 원시 코드를 컴파일하여 실행시킬 때 각 전역변수의 사상 결과가 패더리이트에 의하여 인지될 수 있도록 선언된 전역변수들로 구성되는 헤더 파일을 작성하고 이를 주프로그램에 삽입 시킨다.

### 4.3 객체 및 상호작용 클래스의 설계

패더레이트 설계에서 핵심적인 부분은 각각의 패더레이트가 발행하고 접수해야할 애트리뷰트와 파라미터를 HLA SOM 규격에 맞게 설계하고, 개개의 SOM을 통합하여 패더레이션에서 사용할 수 있도록 하나의 FOM으로 구축하는 작업이다. 이 작업은 많은 시간과 인력이 소요되므로 미국 국방성은 설계 지원을 위한 도구로 OMDT(Object Model Development Tool)를 제공하고 있다[19]. OMDT를 사용하여 객체 클래스의 애트리뷰트 및 상호작용 클래스의 파라미터에 대한 발행과 접수 여부를 개별적으로 지정할 수 있다.

NuclearSim - Attribute Table

| Object | Attribute | Datatype | Cardinality | Units | Resoluti | Accuracy | Accuracy Condition |
|---|---|---|---|---|---|---|---|
| NuclearSimObj | name | any | 1 | | | perfect | always |
| | eng1time_request | double | 1 | | N/ | perfect | perfect |
| | eng1time_response | double | 1 | | | perfect | always |
| | es_simtime | double | 1 | | N/ | perfect | perfect |
| | plant_mode | long | 1 | | | perfect | always |
| | period_pow_double_s | double | 1 | | | perfect | always |
| | var_ov_power_svl | double | 1 | | | perfect | always |
| | pr_pzr_svl | double | 1 | | | perfect | always |
| | temp_pzr_gas_svl | double | 1 | | | perfect | always |
| | temp_pzr_water_svl | double | 1 | | | perfect | always |
| | levl_pzr_svl | double | 1 | | | perfect | always |
| | levl_boiler_svl | double | 1 | | | perfect | always |
| | pr_pzr_gc_svl | double | 1 | | | perfect | always |
| | temp_sg_inlet_svl | double | 1 | | | perfect | always |
| | temp_sg_exit_svl | double | 1 | | | perfect | always |
| | speed_mcp_a_svl | double | 1 | | | perfect | always |
| | speed_mcp_b_svl | double | 1 | | | perfect | always |
| | mode_mcp_a_svl | double | 1 | | | perfect | always |

(그림 8) 객체 클래스의 애트리뷰트 설계

NuclearSim - Parameter Table

| Interaction | Parameter | Datatype | Cardinality | Units | Resoluti | Accuracy | Ac |
|---|---|---|---|---|---|---|---|
| NuclearSimEvent | message | any | 1 | | | perfect | always |
| | ismode_request | long | 1 | | | perfect | always |
| | ismode_response | long | 1 | | | perfect | always |
| | reset_request | long | 1 | | | perfect | always |
| | reset_response | long | 1 | | | perfect | always |
| | snap_request | long | 1 | | | perfect | always |
| | snap_response | long | 1 | | | perfect | always |
| | snapdel_request | long | 1 | | | perfect | always |
| | snapdel_response | long | 1 | | | perfect | always |
| | speed_request | any | 1 | | | perfect | always |
| | speed_response | long | 1 | | | perfect | always |
| | back_request | long | 1 | | | perfect | always |
| | back_response | long | 1 | | | perfect | always |
| | bktime_request | long | 1 | | | perfect | always |
| | bktime_response | long | 1 | | | perfect | always |
| | mfi_id | long | 1 | | | perfect | always |

(그림 9) 상호작용 클래스의 파라미터 설계

또한 각 패더레이트에 필요한 SOM을 개별적으로 설계하지 않고 통합된 하나의 FOM으로 직접 설계할 수 있다.

본 연구는 OMDT 개발도구를 사용하여 <표 5>의 전역변수 클래스 사상과 할당 결과를 FOM으로 설계하였다. (그림 8)은 동특성 엔진, 강사, 운전원에 해당하는 각 패더레이트가 발행과 접수에 사용할 객체 클래스의 애트리뷰트 설계 결과의 일부이며, (그림 9)는 상호작용 클래스의 파라미터 설계 결과의 일부이다.

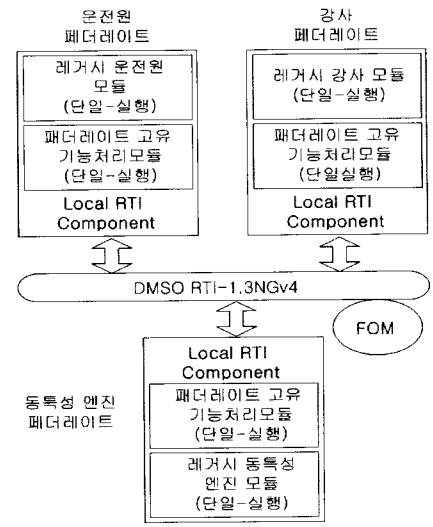
### 4.4 패더레이션의 구성과 실행 구조

본 연구의 패더레이션은 (그림 10)과 같이 운전원, 강사, 동특성 엔진의 세 개 패더레이트로 구성된다. 각 패더레이트의 세부 구성은 앞에서 언급한 (그림 7)과 같다. 또한 각 패더레이트는 한 대의 컴퓨터에서 두 개의 단일 실행을 가지며 이 두 단일 실행은 발행과 접수로 정의된 공유메모리를 통하여 상호간에 데이터를 교환한다.

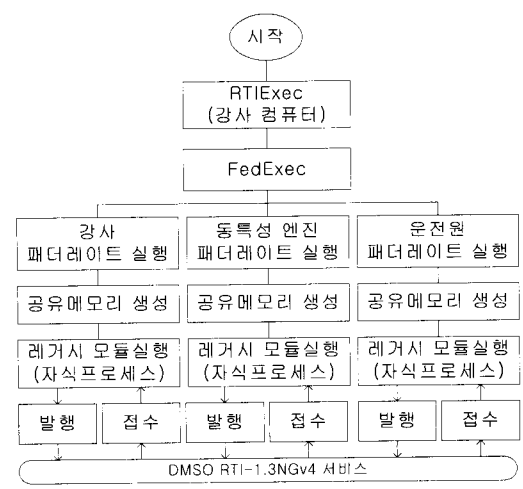
세 개의 패더레이트는 패더레이션이 실행되는 과정에서 (그림 8) 및 (그림 9)와 같이 설계한 FOM을 참조하면서 RTI를 통하여 상호 관심 있는 데이터를 교환한다. 본 연구에서 사용한 RTI는 미국 국방성이 개발한 RTI-1.3NGv4이다[19].

패더레이트들은 레거시 시뮬레이션 환경에서 사용하던 인텔 팬티엄 1.7GHz CPU, 512 MB 메모리와 마이크로소프트웨어 Windows 2000 운영체제 환경을 그대로 사용한다. 동특성 엔진 패더레이트의 레거시 동특성 엔진 모듈은 (그림 6)과 같은 방식을 사용하여 원시코드의 전역변수 부분을 발행과 접수 공유메모리 블록으로 선언하고 Compaq Visual Fortran 6 컴파일러로 컴파일하여 단일 실행 가능한 수정된 레거시 시뮬레이션 코드를 생성하였다. 강사 및 운전원 모듈은 원시 코드가 Visual C++ 6.0으로 작성되어 있다. 따라서 전역변수들을 발행과 접수로 분할한 공유메모리 블록을 선언하고 컴파일하여 단일 실행 코드를 생성하였다. 각각의 패더레이트에는 패더레이트 실행에 필요한 Federate Ambassador 및 RTIAmbassador의 클래스와 RTI API를 추가시킨 후 단일 실행이 가능하게 컴파일 시켰다.

패더레이트가 패더레이션에 참여하여 실행되려면 HLA 규칙사양에 따라 RTIExec(RTI Execution)가 먼저 실행되고 있어야 한다. 본 연구의 경우에는 강사 패더레이트가 시뮬레이션의 전반적인 실행을 통제하므로 강사 패더레이트가 위치한 컴퓨터에서 RTIExec를 실행시킨다. 따라서 (그림 10)의 강사 패더레이트의 LRC는 패더레이션이 실행중인 동안에 CRC(Central RTI Component)의 역할을 담당하게 된




(그림 10) 패더레이션의 구성




(그림 11) 패더레이션 실행과정

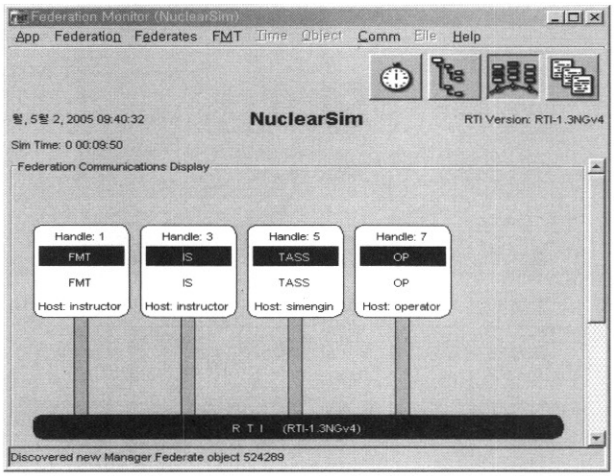
다. RTIExec가 성공적으로 실행되면 RTI는 FedExec(Federation Execution)을 자동으로 실행시키며 패더레이트가 조인(Join)할 수 있도록 대기한다.

각 컴퓨터에서 강사, 동특성 엔진, 운전원 패더레이트를 실행시키면 해당 패더레이트들이 패더레이션에 조인한다. 조인된 각 패더레이트의 고유 기능처리 모듈은 부모 프로세스가 되어 공유메모리를 생성하고 자식 프로세스로써 레거시 시뮬레이션 모듈을 실행시킨다. 이후부터 공유메모리를 통한 전역변수가 상호관심 있는 패더레이트들 간에 RTI를 경유하여 발행 및 접수된다. (그림 11)은 패더레이션의 실행 과정이다.

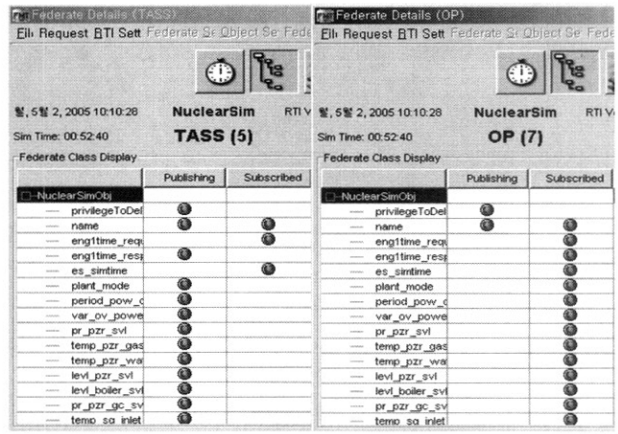
## 5. 패더레이션 실행 결과

HLA 패더레이션은 필요한 데이터만 송신 및 수신하는 다중방송 통신 방식을 사용한다. 따라서 DIS 통신 방식을 사용하는 레거시 시뮬레이션 소프트웨어의 HLA 통합 결과를 통신 트래픽의 감소 관점에서 비교하는 것은 큰 의미를 갖지 못한다. 미국 국방성은 개발된 패더레이션의 실행 상태를 관찰할 수 있는 FMT(Federation Management Tool)를 제공하고 있다. 패더레이션에서 FMT가 실행될 때 FMT 그 자체는 하나의 패더레이트로 패더레이션에 참여하지만 시뮬레이션의 실행에는 아무런 영향을 주지 않으며, 패더레이트 간에 상호교환 되는 객체와 상호작용에 대한 진단정보(발행, 접수, 갱신, 시뮬레이션 진행 속도 등의 상태)를 시각적으로 표시한다. 본 연구는 FMT를 사용하여 관측한 진단정보를 토대로 HLA로 통합한 원자력 시뮬레이션 코드(이하 NuclearSim으로 표기)의 실행 결과를 논한다.

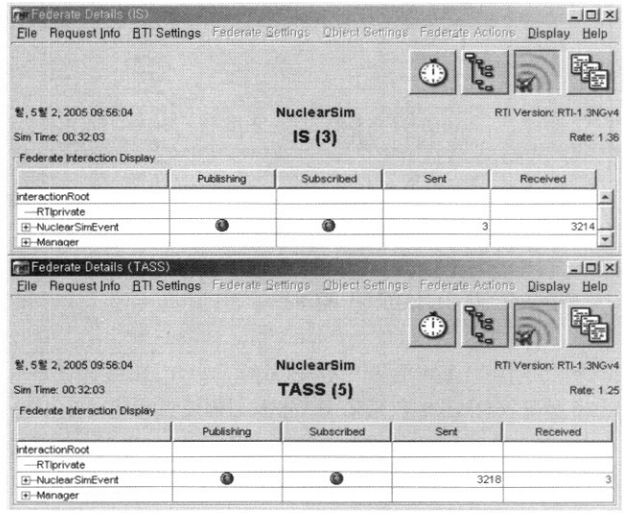
(그림 12)는 이미 개발되어 사용 중인 동특성 엔진 모듈, 운전원 모듈, 강사 모듈이 (그림 7)의 패더레이트 구조에 부합되게 설계 및 구현되어 NuclearSim를 구성하는 세 개의 패더레이트로써 패더레이션에 성공적으로 참여하고 있음을 보인다.




(그림 12) 패더레이트의 패더레이션 참여 상태



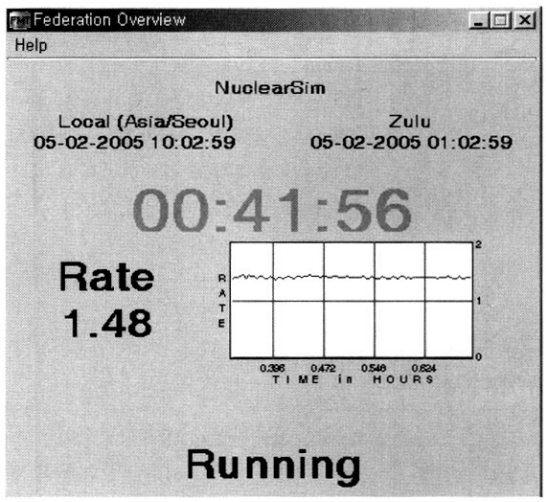

(그림 13) 객체 애트리뷰트의 발행과 접수 상태




(그림 14) 상호작용 파라미터 발행과 접수 상태

(그림 13)은 동특성 엔진 및 운전원 패더레이트의 객체 애트리뷰트에 대한 전역변수 발행과 접수 여부에 대한 진단정보의 일부를 보인 것이다. 이 진단정보를 검토한 결과 <표 3>에 따라 전역변수에 대응하는 애트리뷰트를 사상하고 할당한 설계 결과가 패더레이션 실행 시에 올바르게 동작함을 확인할 수 있었다. 강사 패더레이트의 경우 또한 올바르게 발행과 접수가 동작함을 확인하였다.

(그림 14)는 강사 패더레이트에서 시뮬레이션 초기조건 선택, 원자로를 정지시키는 오동작 주입, 시뮬레이션 시작의 세 가지 강사 명령을 사용하고, NuclearSim이 실행되는 동안에 상호작용 클래스의 파라미터가 강사 및 동특성 엔진 간에 발행(update send) 및 접수(Reflected)되는 상황을 관측한 결과이다. 상호작용의 정확한 동작 여부 확인을 위하여 시뮬레이션이 진행되는 동안에 운전원은 아무런 운전 제어행위도 수행하지 않도록 하였다. (그림 14)에서 보여주듯이 강사의 세 가지 명령에 해당하는 파라미터는 정확하게 동특성 엔진 패더레이터에 전달되었다. 또한 원자로 정지 시뮬레이션의 진행 과정에서 동특성 엔진이 발행하는 상호작용




(그림 15) 패더레이션의 실행 속도 비율

파라미터의 누적 변화 숫자(그림 14 하단의 Sent 항)는 강사 패더레이트가 접수하는 파라미터의 누적 변화 수(그림 14 상단의 Received 항) 보다 항상 큰 값을 보여 상호작용의 파라미터 발행과 접수가 정상 동작함을 알 수 있다.

(그림 15)는 NuclearSim이 1 초의 실시간 실행시간 제약조건 아래서 세 개의 패더레이트가 상호 관심 있는 데이터를 발행 및 접수하면서 시뮬레이션이 진행되는 실행 속도 비율이다. 결과는 최소 약 1.25배에서 최대 1.5배 범위의 실행 속도 향상 범위 내에서 실시간 시뮬레이션이 지속으로 실행 가능함을 보여, 본 연구가 제안하는 발행과 접수 개념의 공유메모리를 사용한 패더레이트 구조 및 레거시 시뮬레이션 코드의 HLA 통합 방법이 현실 환경에서 실용적으로 활용 가능함을 보인다.

## 6. 결 론

본 연구는 레거시 시뮬레이션 코드의 전역변수를 발행과 접수의 공유메모리 블록으로 구분하고, 이를 객체 클래스의 애트리뷰트와 상호작용 클래스의 파라미터로 사상시킨 HLA 패더레이트 통합 방법 및 구조를 제안하였다. 본 연구가 제시하는 방법은 레거시 시뮬레이션 원시 코드의 수정을 최소화 시키면서 적은 노력으로 HLA 패더레이션으로 통합할 수 있는 특징을 갖는다. 단일 컴퓨터상에서 수정된 레거시 시뮬레이션 코드는 하나의 단일 실행 형태이며, RTI 서비스를 이용하기 위하여 원시 코드를 추가한 또 다른 단일 실행인 패더레이트 코드와 공유메모리를 통하여 상호 전역변수 값을 발행 및 접수한다.

원자력발전소 설계와 건설에 사용되는 레거시 시뮬레이션 코드에 본 연구에서 제안하는 방법과 패더레이트 구조를 적용하여 HLA 패더레이션 통합을 시도하였다. FMT를 사용하여 패더레이션의 진행과정을 관측한 결과는 전역변수에 대응하는 객체 클래스의 애트리뷰트 및 상호작용 클래스의 파라미터가 상호 관심을 갖는 패더레이트에게 올바르게 발행 및 접수되고 있음을 보였다. 또한 1초의 실행시간 제약조건을 갖는 레거시 원자력 시뮬레이션 코드가 HLA 패더레이션으로 통합되어 약 1.25배에서 1.5배의 실시간 실행 속도 향상 효과를 갖는 것으로 나타났다.

레거시 원자력 시뮬레이션 코드가 HLA 패더레이션으로 통합된 결과, 시뮬레이션 종료 후 시뮬레이션 진행 과정을 추후에 다시 관측 및 분석하기 위한 재생(Replay) 기능을 사용할 수 없는 문제가 발생하였다. 이는 시뮬레이션이 진행되는 과정에서 HLA가 값이 변경된 애트리뷰트와 파라미터만을 대상으로 발행 및 접수 서비스를 수행하기 때문에 야기되는 문제이다. DIS 통신 방법을 사용하는 기존 시뮬레이션 코드에서는 이러한 문제가 발생하지 않는다. 따라서 본 연구가 제안하는 패더레이트 구조에 재생 기능을 가능하게 하는 추가 연구가 필요하다.

## 참 고 문 헌

[1] Pauline A. Wilcox, Albert G. Burger, and Peter Hoare, "Advanced distributed simulation : a review of developments and their implication for data collection and analysis," Simulation Practice and Theory 8, pp.201-231, 2000.

[2] IEEE Standard 1516-2000 'Standard for Modeling and Simulation(M&S) High Level Architecture', September, 2000.

[3] Don Brutzman, Michale Zyda, J. Mark Pullen, and Katherine L. Morse, "XMSF Challenges for Web-Based Modeling and Simulation : Finding and Recommended Report," Technical Challenges Workshop, Strategic Opportunities Symposium, October, 2002.

[4] U.S. Department of Defense, 'High Level Architecture Rules, Version 1.3', April, 1998.

[5] U.S. Department of Defense, 'High Level Architecture Interface Specification', Version 1.3, April, 1998.

[6] U.S. Department of Defense, 'High Level Architecture Object Model Template Specification, Version 1.3', April, 1998.

[7] Frederick Kuhl, Richard Weatherly, and Judith Dahmann, 'Creating Computer Simulation Systems - An Introduction to the High Level Architecture', Prentice Hall PTR, 1999.

[8] Sean Murphy, "On the Integration of Legacy Biomedical Simulations into Federations using the High Level Architecture," Proceedings of the Fall Simulation Interoperability Workshop, 03F-SIW-104, 2003.

[9] Elizabeth L. White and J. Mark Pullen, "Adapting Legacy Computational Software for XMSF," Proceedings of the Fall Simulation Interoperability Workshop, 03F-SIW-112 2003.

[10] Sean Murphy, James Coolahan, and Robert Lutz, "Human Physiology Simulation Integration Using the HLA : ExerFed1516," Proceedings of the Spring Simulation Interoperability Workshop, 03S-SIW-092, 2003.

[11] Sean Murphy and James Coolahan, "Integrating Cardiac and

Cardiovascular Simulations Using the HLA," Proceedings of the Spring Simulation Interoperability Workshop, 02S-SIW-012, 2002.

[12] Hakan Savasan, Ildeniz Duman, and Musttafa Dinc, "Migration a Legacy Simulation to HLA : Lessons Learned Integrating with New Native HLA Simulations," Proceedings of the Fall Simulation Interoperability Workshop, 03F-SIW-118, 2003.

[13] 김대석, 배종환, 류재철, "HLA 패더레이트 개발을 위한 ROM 프레임워크 설계 및 구현", 정보처리학회논문지 D, 제9-D권 제6호, pp.1137-1144, 2002. 12.

[14] 정재경, 김호정, 원강연, 김종룡, "내장형 시스템을 위한 HLA 기반 분산 실시간 시뮬레이션 환경 구현", 한국정보과학회 2003 춘계학술대회, 제30권 제1호, 2003. 4.

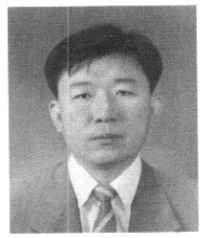
[15] 한관호, 박근옥, 임종태, "미션 크리티컬 시스템에서 실시간 처리를 위한 프로세스 관리 방법", 제 18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호, 2002. 11.

[16] 박근옥, "공유메모리 변수 기반의 CNS 응용 소프트웨어 구조", 제 18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호, 2002. 11.

[17] 박근옥, 임종태, 한관호, "IEEE 1516 기반의 NTS 개발을 위한 권장요건", 한국시뮬레이션학회 추계학술대회 논문집, 2003.

[18] 김희철, 정영종, 임홍식, 양수형, "TASS/SMR 열수력 모델 기술서", KAERI/TR-1835 /2001, 한국원자력연구소

[19] https://sdc.dmso.mil/



**박 근 옥**

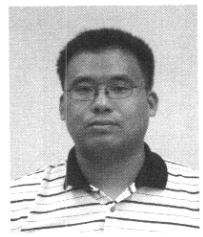
e-mail : gopark@kaeri.re.kr

1991년 서울산업대학교 전자계산학과(학사)

1993년 충남대학교 전자계산학과(석사)

2005년 공주대학교 컴퓨터공학과 박사수료

관심분야 : 정보검색, 시뮬레이션



**한 관 호**

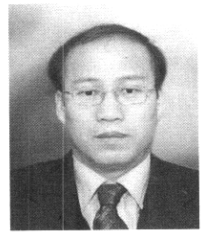
e-mail : knu1445@lycos.co.kr

1996년 공주대학교 전자계산학과(학사)

1998년 공주대학교 컴퓨터공학과(석사)

2001년 공주대학교 컴퓨터공학과 박사수료

관심분야 : 데이터베이스, 시뮬레이션



**임 종 태**

e-mail : jtlim@kongju.ac.kr

1985년 전남대학교 계산통계학과(학사)

1987년 한국과학기술원 전산학과(석사)

1992년 한국과학기술원 전산학과(박사)

1993년~현재 공주대학교 컴퓨터공학부 교수

관심분야 : 정보검색, 시뮬레이션, 바이오인포매틱스