Java/RTI를 위한 Level One Test Procedures 구현

이정욱*, 김용주*, 김영찬**

Implementation of Java/RTI Level One Test Procedures

Jung-uk Lee, Yong-joo Kim, Young-chan Kim

Abstract

HLA (High Level Architecture) is the object, time, and interface standard that proposed for distribution simulation in the US Department of Defense. The HLA is defined by three components: Rules, the HLA Interface Specification, and the Object Model Template (OMT). The RTI (Run-Time Infrastructure) software implements the interface specification. It provides services to simulation applications. To test whether a RTI software is suitable for the standard and all service was implemented is performed through two phases of processes proposed by DMSO. In this paper, we implement Level One Test Procedures of DMSO and apply to NetCust's RTI software. The experimental results are discussed.

Key Words: HLA, RTI, Interface Specification, Level One Test Procedures

* 한밭대학교 정보통신전문대학원 컴퓨터공학과 석사과정
* 한밭대학교 정보통신전문대학원 컴퓨터공학과 석사과정
** 한밭대학교 정보통신전문대학원 컴퓨터공학과 부교수

1. 서론

최근 들어 분산 시뮬레이션을 위해 미 국방성에서 제안한 HLA를 산업계에 적용하기 위한 미들웨어 개발이 활발한데, HLA(High Level Architecture)는 미 국방성에서 제안한 분산 시뮬레이션을 위한 실시간 분산 데이터 공유를 위한 객체, 시간, 인터페이스 규정을 말한다.[1] 이 HLA는 (1)Rules, (2)Interface Specification, (3)Object Model Template 등의 3가지 구성요소로 정의된다.[2][3][4][5] RTI (Run-Time Infrastructure) 소프트웨어는 인터페이스 명세를 구현한 것으로 HLA의 대표 산출물 중에 하나이다. RTI는 분산 운영체계가 응용 프로그램에게 서비스를 제공하는 것과 유사한 방법으로 Federation을 구성하고 있는 Federate들에게 상호연동에 필요한 서비스를 제공한다. 모든 서비스들이 표준에 적합하게 구현되었는지 테스트하기 위한 방법은 DMSO에서 제안한 RTI를 검증하기 위한 2단계 과정을 통해 테스트된다.[5]

RTI 검증은[6][7] 표준과의 비교처리 과정으로서 일반 HLA 사용자 커뮤니티로 DMSO에 의해 제공되는 서비스다. 검증은 표준과의 비교 과정으로써 RTI 검증에 사용되는 표준은 HLA 인터페이스 명세의 특정 버전과 미 국방부에서 지정받은 특정 HLA 인터페이스 명세로 구성된다.[8][9] 인터페이스 규정은 federate으로 RTI에 의해 제공되는 서비스와 RTI로 federate에 의해 제공되는 서비스로 기술된다. 서비스들은 그들의 매개변수, 원문 기술, 상태도, 전제 조건과 포스트조건에 의한 규정으로 기술된다. 이 정보는 테스트 동안 예상되는 RTI 행동을 결정하기 위해 사용된다. 모든 서비스들이 표준에 적합하게 구현되었는지 테스트하기 위한 방법은 DMSO에서 제안한 RTI를 검증하기 위한 2단계 과정을 통해 테스트된다. 1단계는 RTI 개발자에 의해 그들 제품이 2단계 시험에 대한 준비가 되어 있는지 결정하기 위한 자가 진단을 할 수 있으며, 이 테스트는 HLA 인터페이스 명세서의 모든 중요한 영역에서 단순히 RTI의 능력을 보여 주고, 각 테스트는 단계별 명령과 예상되는 결과로 구성된다. 2단계 시험은 RTI 검증 기관에 의해 제공될 수 있다. 검증을 위한 요청서와 소프트웨어를 제출하면, 후보 RTI가 HLA 인터페이스 명세서에서 지정 받은 모든 요구 사항을 고수하는지 확인한다.

본 논문에서는 (주) 넷커스터마이즈가 HLA Spec 1.3에 기반 하여 Java 언어로 개발한 미들웨어인 Java/RTI를[10] 테스트하기 위한 1단계 테스트 절차와 그 방법에 대해 논한다. 테스트 결과를 토대로 각 영역에서 DMSO에서 제안한 Level One Test Procedures에 따라 RTI가 Interface Specification에 적합하게 구현되었는지 확인하며, 각 단계마다 올바른 명령과 예상된 결과가 나오는지 검증한다.

2. HLA/RTI 개요

2.1 HLA

HLA(High Level Architecture)는 분산 시뮬레이션 모델들간의 상호 운용성을 높이기 위해 미 국방성에서 제안된 객체, 시간, 인터페이스 규정을 말하며 3가지 구성요소로 정의된다.

- Rules
  - Federation에 있는 시뮬레이션의 적절한 상호작용을 보장
  - 시뮬레이션과 Federate의 책임/권한 기술
- Interface Specification
  - RTI가 제공하는 서비스들을 정의
  - 각 Federate이 제공하는 "호출(callback)" 함수들을 확인
- Object Model Template(OMT)
  - 기록 정보에 대한 공통양식(방법)을 제공
  - 모델은 Federation Object Model(FOM), Simulation Object Model(SOM), Management Object Model(MOM)

2.2 RTI

RTI는 HLA의 구성요소 중에서 Interface Specification을 구현한 것으로써 Federation을 구성하고 있는 Federate들에게 상호연동에 필요한 서비스들을 제공한다. RTI 서비스는 Federation의 구축과 소멸을 용이하게 하며 Federate간의 객체 선언과 관리를 지원하고 Federation의 시간 관리를 도와주며 Federate의 논리 집단에게 효율적인 통신을 제공한다. RTI는 3개의 주요 구성요소로 이루어져 있으며 6개의 서비스를 제공한다.

- RiExec - RTI 실행
  - 하나의 컴퓨터 기종에서 실행
  - 다수의 Federation 실행을 관리
  - 사용자 명령문에 대한 인터페이스 콘솔
- FedExec - Federation 실행
  - 실행중인 Federation마다 하나씩 존재
  - 다수의 Federate을 관리
  - 사용자 명령문에 대한 인터페이스 콘솔
- libRTI - RTI 라이브러리
  - HLA 서비스들을 Federate에게 확장시키는 방법

<표 1> RTI 서비스

서비스	기능
Federation Management	- Federation의 생성과 소멸 - Federate의 참여 및 탈퇴
Declaration Management	- 분산 객체의 정의 - 객체에 대한 관심처리
Object Management	- 객체와 상호작용의 생성과 삭제, 변경을 처리 - 객체 등록 및 배포 관리
Ownership Management	- 소유권의 이전 처리 및 소유권 공유 관리
Time Management	- 시뮬레이션의 시간 설정 - 동기화 및 시간 처리
Data Distribute Management	- 데이터 공유를 위한 효율적인 라우팅 지원 - 라우팅의 생성 및 소멸 - 라우팅 조건의 설정 및 관리

3. Level One Test Procedures

RTI가 Interface Specification에 적합하게 구현되었는지 테스트하기 위해서는 각 영역에서 서비스 호출에 대한 예측된 결과가 정확히 나오는지 확인함으로써 살펴볼 수 있다. 현재 DMSO는 1단계 테스트에 대한 특별한 소프트웨어 없이 단지 테스트 프로시져와 .FED 화일만 제공하고 있다. DMSO의 Level One Test Procedure는 각 영역을 8개로 나누고 각 호출 서비스에 대한 예측결과로써 구성 되어 있다.

1단계 테스트는 그들의 제품이 2단계 테스트에 대한 준비가 완료되었을 때 RTI 개발자에 의해 수행될 수 있다. RTI 1.3을 사용하는 모든 테스트들은 Federation Execution Data file(FED)을 사용하고, RTI 1516을 사용하는 모든 테스트들은 Federation Object Model Document Data(FDD)를 사용한다. 이 테스트를 위해 각 Federate은 하나의 문자로 배정된다: "A", "B". 또한 두 종류의 문자가 테스트에서 발견된다: "행동", "관찰". "행동" 은 각 Federate에서 서비스의 이름을 호출함으로써 행해지고, "관찰"은 서비스 호출로 발생할 수 있는 예상 결과를 관측하는 것이다. 서비스 호출에서 명백한 상황을 제외하고는 예외처리가 나와서는 안된다.

3.1 Joining Federation

하나의 Federation은 Federate이 Federation에 조인하기 전에 존재해야 하고 모든 Federate들은 Federation이 소멸되기 전에 하

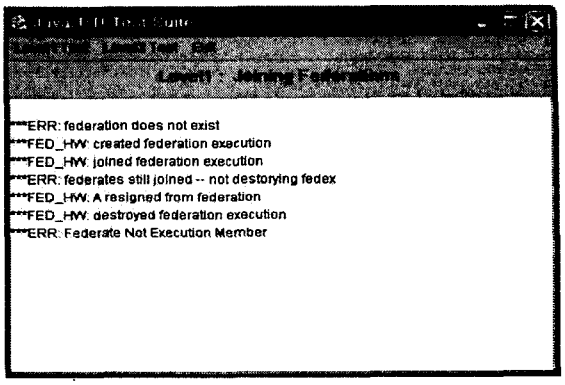
나의 Federation으로부터 탈퇴되어야 한다. Federate은 탈퇴하기 전에 하나의 Federation에 조인되어야 한다.

<그림 1>은 DMSO의 Level One Test Procedures의 Joining Federation에 관련된 내용이다. Federation Management는 Federation 생성, Federate 조인, Federation으로부터의 Federate의 탈퇴 및 Federation의 소멸 등에 관한 RTIambassador 서비스 호출에 대한 내용이다.

	Federate	Step
1	A	Invoke Join Federation Execution("type1","Verification")
2	A	Observe that the service invocation was unsuccessful giving exception "The specified federation execution does not exist"
3	B	Create Federation Execution("Verification","LevelOne.fed")
4	A	Join Federation Execution("type1","Verification")
5	B	Invoke Destroy Federation Execution("Verification")
6	B	Observe that the service invocation was unsuccessful giving exception "Federates are joined to the federation execution"
7	A	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
8	B	Destroy Federation Execution("Verification")
9	B	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
10	B	Observe that the service invocation was unsuccessful giving exception "The federate is not a federation execution member"

<그림 1> Joining Federations

<그림 2>에서 보는 봐와 같이, 처음의 에러는 Federation이 생성되기도 전에 조인을 시도함으로써 나타나는 것으로 이미 프로시져에서 예상했던 결과가 나온다는 것을 볼 수 있다. 두 번째 에러는 Federate들이 여전히 Federation에 조인되어 있는데 Federation을 소멸시키려고 해서 나오는 에러고, 마지막은 Federation이 이미 소멸되었는데 탈퇴 함수를 부름으로써 발생하는 에러이다. 이처럼 모든 테스트에서 서비스 호출에 대한 예상된 결과를 관찰함으로써 테스트를 수행하게 된다.



<그림 2> Joining Federations 실행 결과

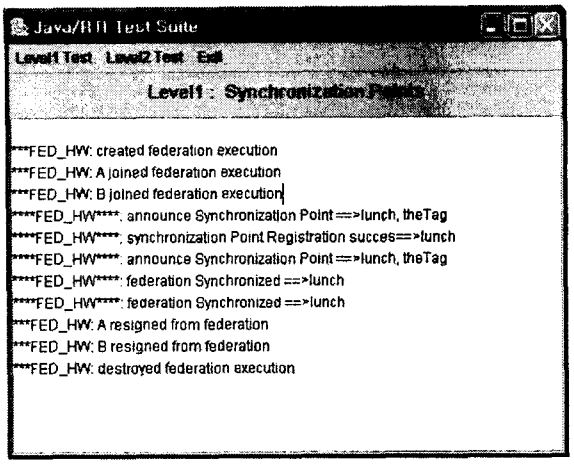
3.2 Synchronization Points

적절한 동기화 라벨이 등록되면 모든 조인된 Federate들에게 알려주게 된다. 라벨에 대한 Federation 동기화는 마지막으로 조인된 Federate이 라벨을 획득한 이후에 발생하게 된다.

	Federate	Step
1	A	Invoke Create Federation Execution("Verification", "LevelOne.fed")
2	A,B	Invoke Join Federation Execution("type1", "Verification")
3	A	Invoke Register Federation Synchronization Point("lunch", "theTag")
4	A	Observe a successful Confirm Synchronization Point Registration("lunch")
5	A,B	Observe Announce Synchronization Point("lunch", "theTag")
6	A	Invoke Synchronization Point Achieved("lunch")
7	B	Invoke Synchronization Point Achieved("lunch")
8	A,B	Observe Federation Synchronized("lunch")
9	A,B	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
10	A	Invoke Destroy Federation Execution("Verification")

<그림 3> Synchronization Points

<그림 1>에서의 호출은 Federateambassador의 콜백(callback)이 없는 단지 RTIambassador 서비스 호출에 대한 것이었다. 그러나 <그림 3>에서는 RTIambassador 서비스 호출에 대해 콜백 함수가 불려지는 구조로 되어있다.



<그림 4> Synchronization 실행 결과

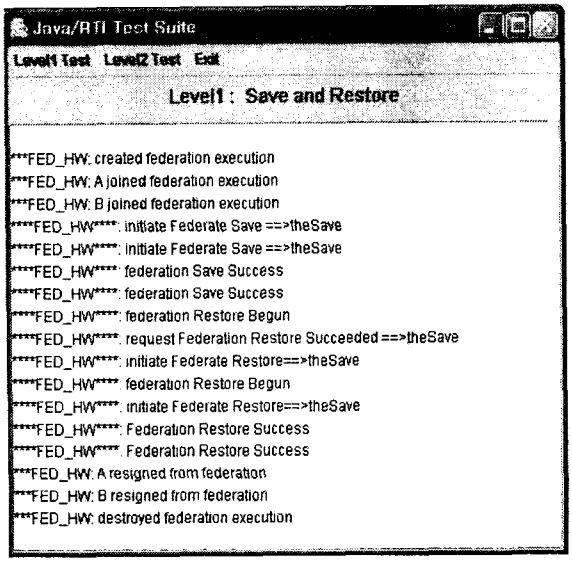
실행 결과에서 보듯이 <그림 3>의 3번 호출에 의한 결과로 4번과 5번의 콜백 함수가 불려지고, 6, 7번의 호출로 8번의 콜백 함수가 불려짐을 알 수 있다.

3.3 Save And Restore

	Federate	Step
1	A	Invoke Create Federation Execution("Verification", "LevelOne.fed")
2	A,B	Invoke Join Federation Execution("type1", "Verification")
3	A	Invoke Request Federation Save("theSave")
4	A,B	Observe Initiate Federate Save("theSave")
5	A,B	Invoke Federate Save Begun()
6	A,B	Invoke Federate Save Complete(Success)
7	A,B	Observe Federation Saved(Success)
8	B	Invoke Request Federation Restore("theSave")
9	B	Observe Confirm Federation Restoration Request("theSave", Success)
10	A,B	Observe Federation Restore Begun() and then Initiate Federate Restore("theSave", federateHandle)
11	A,B	Invoke Federate Restore Complete(Success)
12	A,B	Observe Federation Restored(Success)
13	A,B	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
14	A	Invoke Destroy Federation Execution("Verification")

<그림 5> Save and Restore

Federation 저장 라벨은 즉시 모든 조인된 Federate들에게 제공되어야 한다. Federation 저장의 완료는 각자의 저장이 성공적으로 완료됐다고 양쪽의 Federate들에게 알려줄 때 선언된다. 일단 Federation 저장이 성공적으로 완료되면, 그것은 Federation 복귀에도 사용될 수 있다.



<그림 6> Save and Restore 실행 결과

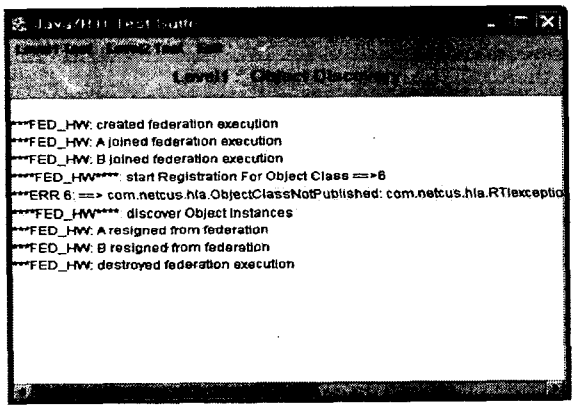
3.4 Object Discovery

객체 관리는 객체의 생성 측면에서는 인스턴스의 등록과 갱신을, 객체의 소비 측면에서는 인스턴스의 발견과 참조를 포함한다. 또한, 상호작용의 송/수신과 관련된 방법 및 소비자 요구에 근거한 인스턴스의 갱신을 조절하고, 기타 다른 여러 가지 지원함수들을 포함한다. 하나의 Federate은 어떤 다른 Federate이 클래스나 수퍼 클래스를 subscribe할 때 객체 인스턴스의 등록을 시작한다. 객체 인스턴스의 발견은 관심을 나타낸 객체 인스턴스가 등록되었을 때 발견할 수 있다. RTIambassador의 register ObjectInstance() 방법은 새로운 인스턴스가 생성되었음을 LRC에게 알린다. 이 방법에는 새로운 객체 인스턴스의 객체 클래스와 하나의 선택된 객체 이름이 요구된다.

	Federate	Step
1	A	Invoke Create Federation Execution("Verification", "LevelOne.fed")
2	A,B	Invoke Join Federation Execution("type1", "Verification")
3	A	Invoke Get Object Class Handle("objectRootLA"). Keep the returned object class handle and refer to it as objectClassB.
4	A	Invoke Get Attribute Handle("X", objectClassB). Keep the returned class attribute handle and refer to it as attributeX.
5	A	Invoke Get Attribute Handle("Y", objectClassB). Keep the returned class attribute handle and refer to it as attributeY.
6	A	Invoke Publish Object Class(objectClassB, {attributeX, attributeY})
7	B	Invoke Get Object Class Handle("objectRootLA"). Keep the returned object class handle and refer to it as objectClassA.
8	B	Invoke Subscribe Object Class Attribute(objectClassA, {attributeX})
9	A	Observe Start Registration for Object Class(objectClassB)
10	B	Invoke Register Object Instance(objectClassA)
11	B	Observe that the service invocation was unsuccessful giving exception "The federate is not publishing the specified object class"
12	A	Invoke Register Object Instance(objectClassB)
13	B	Observe Discover Object Instance(objectClassA)
14	A,B	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
15	A	Invoke Destroy Federation Execution("Verification")

<그림 7> Object Discovery

<그림 7>에서 10번의 호출에 대한 콜백 함수로서 Discover Object Instance()가 발생하지 않는 이유는 Federate가 지정된 객체 클래스를 publish하지 않았기 때문이다. 즉, 관심을 나타낸 객체를 등록할 때 발견할 수 있다.



<그림 8> Object Discovery 실행 결과

3.5 Attribute Divestiture and Acquisition

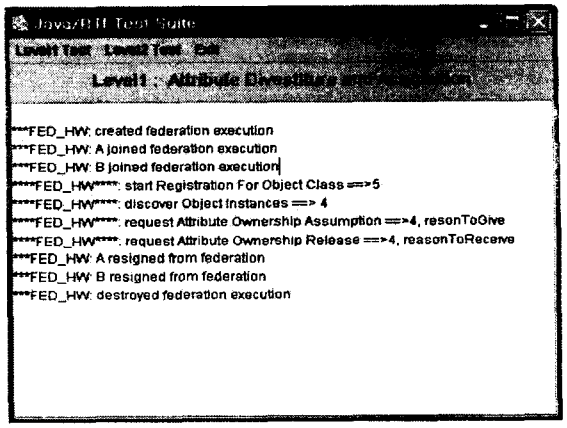
소유권 이관은 한 Federate이 갱신 책임이나 객체 인스턴스에 대한 삭제 권한을 소유하도록 제한하여 다른 Federate에게 소유권을 넘길 때 사용된다. 소유권은 무조건적(unconditionally) 혹은 협의된(negotiated) 형태로 넘겨주게 된다. 무조건적 이관은 갱신과 삭제 책임을 받을 다른 Federate에게 어떠한 위임 없이 갱신과 삭제 책임을 Federate에서 해제한다. 협의된 이관은 새로운 소유자가 확인되고 정식 교환 과정이 완료될 때까지 책임을 보유하는 정식 교환 방법이다.

	Federate	Step
1	A	Invoke Create Federation Execution("Verification", "LevelOne.fed")
2	A,B	Invoke Join Federation Execution("type1", "Verification")
3	A	Invoke Get Object Class Handle("objectRootLA"). Keep the returned object class handle and refer to it as objectClassA.
4	A	Invoke Get Attribute Handle("X", objectClassA). Keep the returned class attribute handle and refer to it as attributeX.
5	A,B	Invoke Publish Object Class(objectClassA, {attributeX})
6	B	Invoke Subscribe Object Class Attribute(objectClassA, {attributeX})
7	A	Observe Start Registration for Object Class(objectClassA)
8	A	Invoke Register Object Instance(objectClassA). Keep the returned object handle and refer to it as theRegisteredObjectHandle.
9	B	Observe Discover Object Instance(objectClassA) Keep the returned object handle and refer to it as theDiscoveredObjectHandle.
10	A	Invoke Negotiated Attribute Ownership Divestiture(theRegisteredObjectHandle, {attributeX}, "reasonToGive")
11	B	Observe Request Attribute Ownership Assumption(theDiscoveredObjectHandle, {attributeX}, "reasonToGive")
12	A	Invoke Cancel Negotiated Attribute Ownership Divestiture(theRegisteredObjectHandle, {attributeX})
13	B	Invoke Attribute Ownership Acquisition(theDiscoveredObjectHandle, {attributeX}, "reasonToReceive")
14	A	Observe Request Attribute Ownership Release(theRegisteredObjectHandle, {attributeX}, "reasonToReceive")
15	A,B	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
16	A	Invoke Destroy Federation Execution("Verification")

<그림 9> Attribute Divestiture and Acquisition

협의된 이관은 무조건적 보다 더 복잡하다. 책임을 넘기고자 하는 Federate은 Negotiated Attribute Ownership Divestiture()를 호출한다. 일부 또는 모든 속성의 발생을 포기할 능력을 가진 Federate은 Request Attribute Ownership Assumption() 호출(callback) 함수를 통해 통지된다. 제공된 하나 이상의 속성을

얻고자 하는 Federate은 Attribute Ownership Acquisition()을 사용한다. Federate이 책임으로부터 벗어나고자 할 때, 이관이 시작된 Federate은 더 이상 등록된 속성에 대한 책임을 가지지 않는다는 Attribute Ownership Divestiture Notification() 호출(callback)을 받게 된다. 그 속성에 대해 책임을 얻게 되는 Federate은 Attribute Ownership Acquisition Notification() 호출(callback) 함수를 통해 그 속성에 대한 새로운 책임 소유를 알게 된다.



<그림 10> Attribute Divestiture and Acquisition 실행 결과

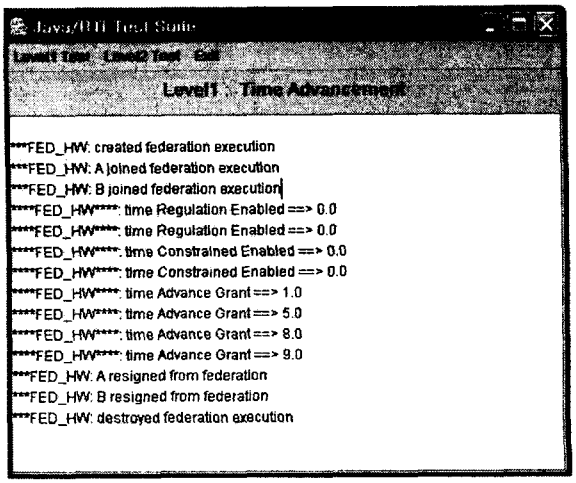
3.6 Time Advancement

HLA는 다양한 시간관리 정책을 가지고 있으며, RTI는 Federate들 간의 사건 교환을 조정하기 위해 선택적 시간관리 서비스를 제공한다. 사건들은 시점과 연관되어 있으며, RTI는 원인과 행위를 보장할 수 있다. 기본적으로 RTI는 Federate간의 시간 조정을 시도하지 않는다. HLA의 장점 중에 하나는 시간관리 정책을 다양하게 지원할 뿐만 아니라, 서로 다른 시간 정책을 가진 Federate 사이에서 상호 운용성을 기대할 수 있다는 것이다. Federation 내에서 시간은 항상 앞으로 진행된다. 일반적으로 시간 진행은 객체 관리 서비스를 통해 조정되어야 하며, 이를 통해 Federate들에게 전달되는 정보는 원인과 결과가 정확해야 하고, 순차적이어야 한다.

	Federate	Step
1	A	Invoke Create Federation Execution("Verification", "LevelOne.fed")
2	A,B	Invoke Join Federation Execution("type1", "Verification")
3	A,B	Invoke Enable Time Regulation (0, 2)
4	A,B	Observe Time Regulation Enabled
5	A,B	Invoke Enable Time Constrained
6	A,B	Observe Time Constrained Enabled
7	A	Invoke Time Advance Request(5)
8	B	Invoke Time Advance Request(1)
9	B	Observe Time Advance Grant(1)
10	B	Invoke Time Advance Request(8)
11	A	Observe Time Advance Grant(5)
12	A	Invoke Time Advance Request(9)
13	B	Observe Time Advance Grant(8)
14	A	Observe Time Advance Grant(9)
15	A,B	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
16	A	Invoke Destroy Federation Execution("Verification")

<그림 11> Time Advancement

Federate은 기본적으로 "규제하지도(regulating) 규제 받지도(constrained)" 않는다. Federate은 규제하거나 규제받는 Federate 요청을 위해 RTIambassador 멤버 함수인 enable Time Regulation()이나 enable Time Constrained() 함수를 이용한다.



<그림 12> Time Advancement 실행 결과

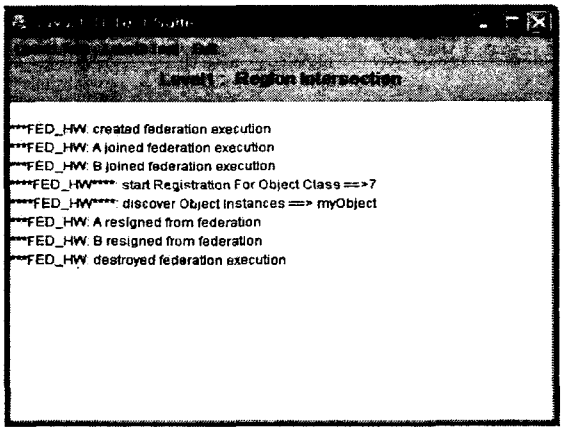
3.7 Region Intersection

Federate은 데이터와 지역(Region)을 관련시킨다. 등록과 참조를 위해 지역이 사용될 때 객체 인스턴스는 참조를 위해 사용되는 지역이 등록을 위해 사용되는 지역과 교차할 때까지 발견되지 않는다.

	Federate	Step
1	A	Invoke Create Federation Execution("Verification", "LevelOne.fed")
2	A,B	Invoke Join Federation Execution("type1", "Verification")
3	A	Invoke Get Object Class Handle("objectRootA.C"). Keep the returned object class handle and refer to it as objectClassC.
4	A	Invoke Get Attribute Handle("Z", objectClassC). Keep the returned class attribute handle and refer to it as attributeZ.
5	A	Invoke Get Routing Space Handle("RS"). Keep the returned routing space handle and refer to it as spaceHandleRS.
6	A	Invoke Get Dimension Handle("D", spaceHandleRS). Keep the returned dimension handle and refer to it as dimensionD.
7	A	Invoke Publish Object Class(objectClassC, {attributeZ})
8	A	Invoke CreateRegion(spaceHandleRS, {(2..4)}) Keep the returned region and refer to it as registrationRegion. The number of intermediate steps required to create a region depends on the API being used. These intermediate steps will include creation of a region object with a single extent, setting the lower bound of the extent's dimension "D" to 2, setting the upper bound of the extent's dimension "D" to 4, and the notification of the RTI that the extent adjustment is complete.
9	B	Invoke CreateRegion(spaceHandleRS, {(5..7)}) Keep the returned region and refer to it as subscriptionRegion. The number of intermediate steps required to create a region depends on the API being used. These intermediate steps will include creation of a region object with a single extent, setting the lower bound of the extent's dimension "D" to 5, setting the upper bound of the extent's dimension "D" to 7, and the notification of the RTI that the extent adjustment is complete.
10	B	Invoke Subscribe Object Class Attribute With Region(objectClassC, subscriptionRegion, {attributeZ})
11	A	Observe Start Registration for Object Class(objectClassC)
12	A	Invoke Register Object Instance With Region(objectClassC, {(attributeZ, registrationRegion)}, "myObject")
13	B	Observe Discover Object Instance() does not occur.
14	A	Invoke Modify Region(registrationRegion, {(2..6)}). The number of intermediate steps required to modify a region depends on the API being used. These intermediate steps will include resetting the upper bound of the extent's dimension "D" to 6 and the notification of the RTI that the extent adjustment is complete.
15	B	Observe Discover Object Instance("myObject", objectClassC)
16	A,B	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
17	A	Invoke Destroy Federation Execution("Verification")

<그림 13> Region Intersection

Federate은 특정 라우팅 공간에 새로운 지역을 구성하기 위해 RTIambassador 방법인 createRegion() 함수를 호출하며, 라우팅 공간은 FED 파일에 선언되어 있어야만 한다. 지역이 국부적으로 수정되면 그 변경내용을 RTI에 전달해야 하는데 RTIambassador 방법인 notifyAboutRegionModification() 함수가 그 역할을 담당한다. 그림 13에서 보면 13번째에서 인스턴스를 발견할 수가 없는데 이는 교차되는 지역이 발생하지 않아서이다. 이에 지역을 수정하여 인스턴스를 발견한다.



<그림 14> Region Intersection 실행 결과

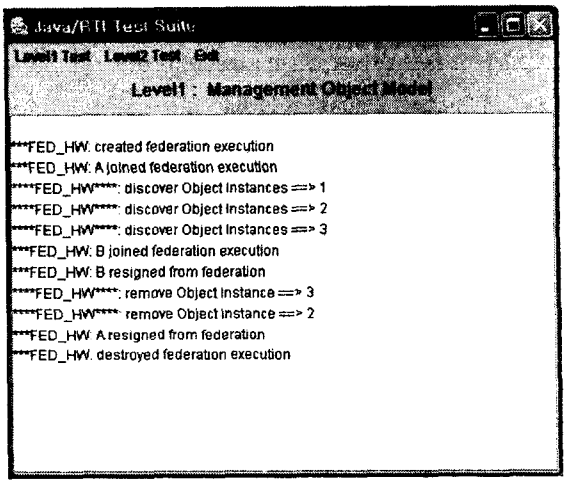
3.8 Management Object Model

객체 관리 모델(MOM)은 RTI와의 이용을 위해 고안된 FED 파일의 FOM 계층구조에 표현되어야 하는 여러 객체와 상호작용 클래스로 구성된다. 이러한 클래스는 Federation을 구성하는 LRC 들의 내/외부 특성에 대한 정보를 얻을 수 있게 하는 메커니즘을 형성한다.

	Federate	Step
1	A	Invoke Create Federation Execution("Verification", "LevelOne.fed")
2	A	Invoke Join Federation Execution("type1", "Verification")
3	A	Invoke Get Object Class Handle("objectRoot.Manager.Federate"). Keep the returned object class handle and refer to it as objectClassFederate.
3	A	Invoke Get Object Class Handle("objectRoot.Manager.Federation"). Keep the returned object class handle and refer to it as objectClassFederation.
4	A	Invoke Get Attribute Handle("FederateType", objectClassFederate). Keep the returned class attribute handle and refer to it as attributeFederateType.
5	A	Invoke Get Attribute Handle("FederationName", objectClassFederation). Keep the returned class attribute handle and refer to it as attributeFederationName.
6	A	Invoke Subscribe Object Class Attributes(objectClassFederation, {attributeFederationName})
7	A	Observe Discover Object Instance(objectClassFederation)
8	A	Invoke Subscribe Object Class Attributes(objectClassFederate, {attributeFederateType})
9	A	Observe Discover Object Instance(objectClassFederate)
10	B	Invoke Join Federation Execution("type2", "Verification")
11	A	Observe Discover Object Instance(objectClassFederate). Keep the returned object handle and refer to it as theOtherFederate.
12	B	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
13	A	Observe Remove Object Instance(theOtherFederate)
14	A	Invoke Resign Federation Execution(Delete Objects and Release Attributes)
15	A	Invoke Destroy Federation Execution("Verification")

<그림 15> Management Object Model

Manager.Federation 객체 클래스는 RTI에 의해 Federation 실행 당 하나씩 만들어지며, Manager.Federate 객체 클래스는 LRC에 의해 Federate 당 하나씩 만들어진다. 객체 클래스 Manager.Federate의 인스턴스는 새로운 Federate이 조인되거나 삭제될 때 RTI에 의해 등록된다.



<그림 16> Management Object Model 실행

4. 결론

본 논문에서는 (주) 넷커스터마이즈가 HLA Spec 1.3에 기반 하여 Java 언어로 개발한 미들웨어인 Java/RTI가 모든 서비스들이 표준에 적합하게 구현되었는지 테스트하기 위한 방법으로 DMSO에서 제안한 RTI를 검증하기 위한 2단계 과정 중 1단계 테스트를 구현해 봤다. 1단계 테스트는 2단계 테스트로 가기 위한 RTI 개발자들에 의해 수행되는 중간 단계라 말할 수 있다. 현재 1단계 테스트에 대한 어떠한 소프트웨어도 나와있지 않지만 본 논문에서 기술된 프로그램을 통해 적합한 수행 결과를 얻는다면 1단계 테스트를 성공적으로 통과했다고 말할 수 있다. 그렇지만 HLA Compliant로서 인증 받기 위해서는 2단계 테스트를 통과해야만 인증을 받을 수 있다. 그렇게 하기 위해 차후 2단계 테스트에 대한 테스트 절차와 Test Suite 개발에 대한 연구가 더 필요하다.

참고문헌

[1] U.S. Department of Defense, "RTI 1.3-Next Generation Programmer's Guide Version 3.2", 28 April 2000

[2] 장상철외 3명, "국방모의분석체계 구축을 위한 상위체계구조[HLA] 기술 연구", 한국국방연구원, 1999, 10

[3] U.S. Department of Defense, "HLA Rules version 1.3", 1998 IEEE Standards Draft, February 5, 1998

[4] U.S. Department of Defense, "HLA Object Model Template Specification version 1.3", 1998 IEEE Standards Draft, April 20, 1998

[5] U.S. Department of Defense, "HLA Interface Specification version 1.3", 1998 IEEE Standards Draft, April 20, 1998

[6] John Tufarolo, James Ivers, Timothy C. Hyon, "Automated Distributed System Testing: Application of an RTI Verifications System", 1999 Winter Simulation Conference

[7] John Tufarolo, James Ivers, Timothy C. Hyon, "Automated Distributed System Testing: Designing an RTI Verifications System", 1999 Winter Simulation Conference

[8] U.S. Department of Defense, "Level One Test Procedure version 1.0"

[9] U.S. Department of Defense, "HLA Interface Specification Test Procedures version 1.3", April 1998

[10] 엄용섭, 박성용, 안명수, "HLA 규격 1.3에 적합한 미들웨어 Java/RTI 개발", 한국시뮬레이션학회 02 추계학술대회논문집, 2002. 11 pp. 53-57

주 작 성 자 : 이 정 욱
논문투고일 : 2003. 09. 28
논문심사일 : 2003. 11. 12(1차), 2003. 11. 17(2차),
2003. 12. 09(3차)
심사판정일 : 2003. 12. 09

● 저자소개 ●

이정욱

2002 한밭대학교 컴퓨터공학과 학사
2002 ~ 현재 한밭대학교 정보통신전문대학원 석사과정
관심분야: 데이터베이스, 미들웨어.

김용주

2002 한밭대학교 컴퓨터공학과 학사
2002 ~ 2003 한국전자통신연구원 위촉연구원
2002 ~ 현재 한밭대학교 정보통신전문대학원 석사과정
관심분야: 분산 시뮬레이션, 데이터베이스, 미들웨어.

김영찬

1981 ~ 1985 아주대학교 전자공학과 학사
1985 ~ 1987 한국과학기술원 전기 및 전자공학과 석사
1985 ~ 1990 삼성전자 ASIC 센터 주임연구원
1990 ~ 1995 한국과학기술원 전기 및 전자공학과 박사
1995 ~ 1996 한국과학기술원 위촉연구원
1996 ~ 1997 Univ. of Arizona, Visiting Scholar
1997 ~ 1998 한국전자통신연구원 선임연구원
1998 ~ 현재 한밭대학교 정보통신전문대학원 컴퓨터공학과 부교수
관심분야: 분산 시뮬레이션, 데이터베이스, 미들웨어, 임베디드 시스템.