에이전트 기술을 사용한 HLA/RTI 기반 시뮬레이션의 구현

김용주* · 김영찬**

An Implementation of Simulation based on HLA/RTI using Agent Technique

Yong-joo Kim · Young-chan Kim,

Abstract

HLA-RTI is middleware for the distribute simulation that developed in the US Department of Defense. This provides fast accomplishment speed and reliability than distribute simulation Middleware by transfer.

However, DDM(Data Distribution Management) service is used as data filtering technology in the existing HLA-RTI. As for this, the problem that network traffic increases in data exchange between the mobility simulation objects is generated, it proposes applying agent technology to the mobility simulation object in order to solve these problems in this paper in this. And this paper applies that to practical simulation and analyzes performance between each data filtering technology with comparison.]

Key Words : HLA, RTI, DDM, Agent, filtering

Ⅰ. 서론

현재 분산 시뮬레이션 환경에서 널리 사용되는 High Level Architecture(HLA)는 미국방부(DoD) 산하 DMSO의 주도 하에 개발된 국방관련 분산/병렬 시뮬레이션을 위한 표준 개발 프레임 워크로서, 응용 프로그램간에 상호 운용성, 재사용성 제고를 목적으로 하고 있다. [1]

HLA의 초기 개발 목적은 국방 관련 응용 분야인 실시간 분산/병렬 시뮬레이션을 지원하는데 있었지만, HLA에서 정의된 서비스는 일반적인 분산 시뮬레이션 시스템(항만, 교통, 항공)에도 적용되고 있다. 그러나 다양한 시뮬레이션 환경에 적용되면서 시뮬레이션 자체가 점점 복잡해지게 되었다. 복잡한 데이터의 교환으로 인하여 네트워크의 처리비용이 증가하였고 데이터 필터링을 위한 처리비용 또한 증가하게 되었다.[2]

본 논문에서는 이러한 문제점을 해결하기 위해 선택적으로 Agent를 사용하는 구조를 제안한다.[2] 제안된 방법을 사용하여 시뮬레이션 객체에 대한 데이터 필터링 알고리즘을 설계하고 구현한다. 또한 기존의 DDM 서비스와 Agent 데이터 필터링 기법에 대한 성능을 비교하며 데이

*한밭대학교 정보통신 전문대학원 컴퓨터공학과 석사과정
**한밭대학교 정보통신 전문대학원 컴퓨터공학과 부교수

터 필터링 기법을 실제적인 시뮬레이션에 포함하여 구현한다.

2. 배경지식

2.1 HLA

HLA는 Rule, OMT, Interface Specification의 3가지 구성요소를 가지고 있다.[1]

2.2. RTI

RTI는 HLA의 3가지 구성요소 중에서 Interface Specification을 실제적으로 구현한 미들웨어로서 전반적인 시뮬레이션을 수행하게 된다. 이러한 RTI는 Federation Management, Declaration Management, Object Management, Ownership Management, Time Management, Data Distribution Management의 6개의 서비스를 제공한다.[1]

〈표 1〉 RTI 서비스의기능

서비스	서비스 기능
Federation Mgt	- Federation 실행의 생성 - 참여 및 탈퇴 - Federation 실행의 삭제
Declatation Mgt	- Publich/Subscribe / Reflect 객체 선언
Object Mgt	- 객체와 상호작용의 생성 - 갱신 및 삭제
Ownership Mgt	- 속성들의 소유권 이양 - 속성들의 소유권 획득
Time Mgt	- 시뮬레이션의 시간 설정 - 동기화 및 시간 수정
Distribution Mgt	- 갱신 및 수신 지역 생성 - 지역변경 및 삭제

2.3 HLA Data Filtering Technique

RTI는 전송되는 데이터의 형식이나 내용물에 대해 상세히 알지 않아도 되는 구조를 가지며 단지 선언된 관심 영역(Interest region)에만 근거하여 데이터의 Publisher(송신측)와 Subscriber(수신측)를 연결하게 된다.

분산된 환경에서 데이터를 효율적으로 전송하기 위하여 HLA는 데이터 필터링 기법을 제공하며 이러한 기법이 DDM (Data Distribute Management) 서비스이다.

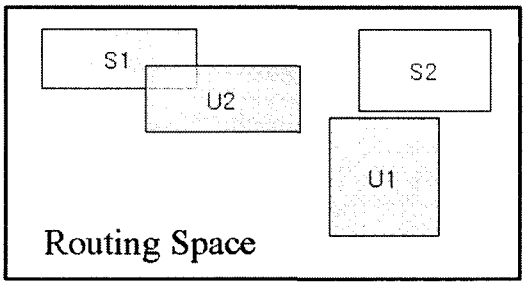
DDM 서비스는 각 Federate들의 관심영역을 위해 유연성 있고, 확장된 메커니즘을 제공하며 이 메커니즘은 RTI의 데이터교환능력을 효율적으로 확장시켜준다.

DDM에서 데이터의 효율적인 분배를 위하여 Routing Space를 선언한다. Routing Space는 Dimension으로 구성되어 있고 또한 Dimension은 각각의 region을 정의한다. region은 extent의 집합으로 정의되며 각각의 extent는 가장 작은 형태의 영역 공간이다.

예를 들어 하나의 시뮬레이션에서 두 개의 Federate가 사용된다면 각각의 Federate1과 Federate2는 자신의 시뮬레이션을 위해 그림 1에서 보는 바와 같은 형태로 Routing Space에 자신의 데이터와 관련된 Update region과(Ui) Subscribe region을(Si)를 지정한다.

각 Federate는 X축과 Y축의 일정한 좌표 형태로 자신의 영역 임계치를 가지며 Federate 1의 Subscribe region인 S1과 Federate 2의 Update region인 U2사이에 서로 겹쳐지는 영역이 발생하게 된다.

이러한 겹쳐진 구간에 있는 데이터만이 두 Federate들 사이에서 교환 되게 된다.



〈그림 1〉DDM 서비스에서의 관심영역

DDM 서비스는 기존의 DM 서비스의 문제점인 Publisher에서 모든 데이터를 Subscriber에게 전송하고 Subscriber는 받은 데이터 중 관련 있는 데이터만은 골라내던 것을 해결하였지만 다음과 같은 문제점이 발생하였다.

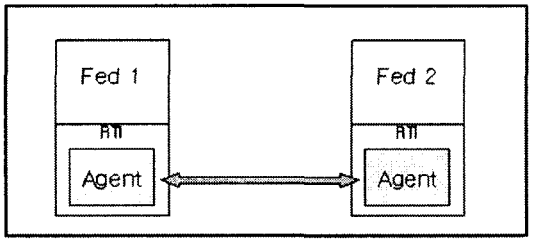
① 이동성이 있는 시뮬레이션 객체의 경우 영역의 지속적인 변경이 필요하므로 많은 계산을 필요로 한다.

② 겹쳐진 구간 내에 존재할 수 있는 관련 없는 데이터의 처리비용이 증가한다.

이러한 문제점을 해결하기 위하여 본 논문에서는 Agent를 사용한 Data Filtering 기법을 제시한다.[2]

3. Agent를 사용한 Data Filtering 기법

Agent를 사용한 RTI는 그림 2와 같은 구조를 가진다.

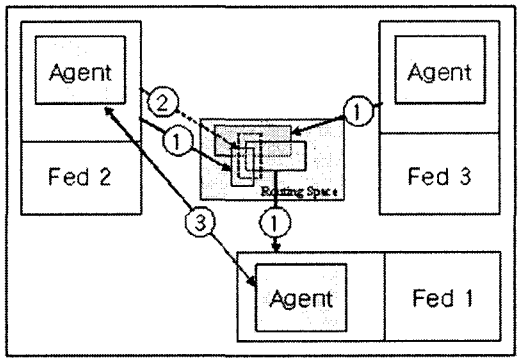


〈그림 2〉 Agnet를 기반으로 하는 RTI

HLA에서 시뮬레이션 객체들은 자신이 관심있는 데이터를 수신하기 위하여 subscribe region을 설정하게 된다. RTI가 일정한 영역을 설정하려고 시도할 때 RTI내에서 컴포넌트 형태로 존재하는 Agent가 이러한 요청을 받아 데이터를 publish하는 Federate의 Agent와 연결되어 데이터를 교환하게 된다. 만일 Update 혹은 subscribe하는 Federate가 region을 변경하였을 경우 Agent간의 파라미터 교환을 통해 그 region에 대한 정보를 계속 유지하게 된다.[2]

Agent를 사용한 데이터필터링 기법은 이동성이 있는 시뮬레이션 객체들 간에는 뛰어난 성능을 발휘하였으나 고정된 시뮬레이션 객체 즉 영역을 지속적으로 변경하지 않는 시뮬레이션에서는 기존의 DDM보다 낮은 성능을 보였다. 또한 많은 시뮬레이션 객체들이 시뮬레이션에서 동작할 때 각각의 시뮬레이션 객체마다 존재하는 Agent를 처리하는 비용이 증가하는 문제점이 발생하였다. 이에 본 논문에서는 이러한 문제점을 해결하는 DDM 서비스를 사용하는 선택적인 Agent의 적용을 제안한다.

그림 3에서 보는 바와 같이 시뮬레이션에서 Fed1은 일정한 Subscribe region을 가지며 Fed2와 Fed3는 일정한 Update region을 가진다. 선택적인 Agent는 다음과 같은 동작을 하게 된다.



〈그림 3〉 DDM에서 선택적인 Agent의 사용

시뮬레이션 초기에는 모든 시뮬레이션 객체가

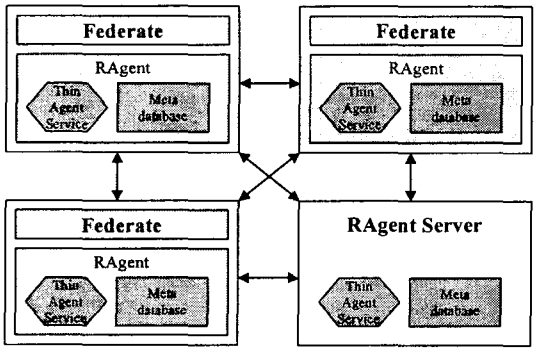
DDM 서비스를 사용하게 된다.

Fed 2가 이동시에 즉 Update region을 변경할 시에 Fed2의 RTI는 Agent에게 데이터 전송을 위임한다. 이때 Fed 2의 RTI는 Routing space내의 관심 영역을 삭제한다. Fed 2의 Agent는 Fed 1의 Agent에 데이터를 전송하게 되며 일정 시간 이상 Fed 2가 이동하지 않을 시 즉 Update region이 변경되지 않을시 Fed 2의 RTI는 Agent의 작동을 중지시키고 DDM내에 영역을 설정하게 된다.

Federate내에 있는 에이전트의 내부 동작은 그림 4에서 보는 바와 같다.

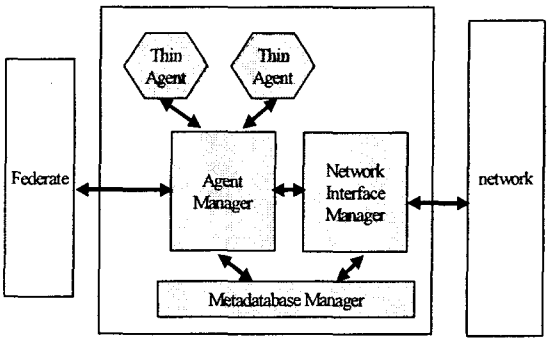
실제적인 Agent의 동작은 Federation 내에 존재하는 Agent의 동작을 관리하는 RAgent Server와 실제적인 시뮬레이션에서 컴포넌트의 형태로 동작하는 RAgent가 있다.

시뮬레이션 내에 존재하는 각각의 Federate내에는 RAgent가 존재하며 자체적으로 관리하는 Metadatabase를 사용하여 Federation내에 참여하는 각각의 federate의 정보를 저장, 관리하게 된다. 각각의 RAgent들은 RAgent 서버와 상호작용하며 전반적인 시뮬레이션 내의 데이터 전송을 담당하게 된다.



〈그림 4〉 Agent 사용시 시뮬레이션의 동작

각각의 RAgent의 내부동작은 그림 5와 같다. 각각의 RAgent내에는 다른 Federate 내에 존재하는 RAgent와 연결된 thin agent가 존재하며 이러한 각각의 thin agent를 관리하는 Agent Manager가 존재한다. Agent Manager는 실제적인 시뮬레이션 코드와 상호작용하며 또한 시뮬레이션에 참여하는 Federate의 정보를 담고 있는 Metadatabase Manager와 네트워크 연결을 담당하는 Network Interface Manager와 연결된다. Network Interface Manager는 Agent Manager와 Metadatabase Manager와 상호 작용하며 네트워크를 사용하여 다른 위치에 있는 simulation과 상호 작용하게 된다.



〈그림 5〉 Client Agent의 내부 동작

이러한 데이터필터링 기법은 모든 시뮬레이션 내에 Agent를 초기화 후 필요시에만 사용하는 방법으로 Agent만을 사용했을 때 발생할 수 있는 고정적인 시뮬레이션 객체의 문제점을 보완할 수 있다.[2] 또한 시뮬레이션 객체에 시간의 흐름에 따라 선택적으로 Agent를 적용할 수 있으므로 많은 수의 객체들이 시뮬레이션에 참여했을 때 Agent의 비용을 줄일 수 있다.

4. 구현

4.1 구현환경

Agent를 사용한 데이터필터링 기법은 Java를 이용하여 구현되었으며 개발환경은 다음과 같다.

〈표 1〉 개발환경

구분	환경
Java	J2SDK 1.4.1
RTI	pRTI 1.3-NG (pitch AB)
OS	Windows XP, Redhat 7.0

Agent를 사용한 데이터필터링기법을 위하여 Agent Manager와 각 federate의 Agent와 데이터를 교환하는 RAgent를 구현하였다. 또한 Agent의 동작과 중지를 수행하는 코드와 실제적인 시뮬레이션 프로그램을 작성하였다.[3] [4] [5]

4.2 RAgent Manager의 구현

RAgent Manager는 Federation에 생성되는 모든 RAgent의 인스턴스를 관리하도록 구현하였다. JavaRMI를 사용하여 물리적으로 분산된 시뮬레이션을 효과적으로 관리하도록 구현하였다.

4.3 RAgent의 구현

RAgent는 다른 Federate들이 데이터를 요청하거나 어떠한 데이터를 다른 Federate로부터 수신하길 원할 때 실제적으로 그 역할을 하는 thin agent의 인스턴스를 생성하여 각 Federate와 일대일의 연결을 생성하게 된다. 또한 여러 개의 thin agent를 관리하는 Agent Manager를 구현하였으며 네트워크 연결을 담당하는 Network Interface manager와 Metadatabase Manager를 구현하였다. 또한 Metadatabase는 Java의 Hash-table 자료형으로 구현하여 수행속도를 증가시켰다.

4.4 시뮬레이션 프로그램의 구현

본 논문에서 Agent를 사용한 데이터필터링기법을 적용하기 위하여 하나의 시뮬레이션을 작성하였으며 프로그램의 동작은 다음과 같다.

① 시뮬레이션 내에는 4개의 컨테이너 선박과 2개의 관제소가 존재한다.

② 2개의 관제소는 일정영역의 레이더를 가지며 레이더 영역 내에서 운항하는 선박의 좌표정보를 지속적으로 수신한다.

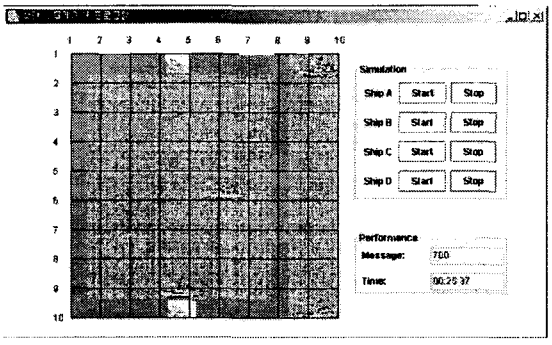
③ 4개의 배들은 좌표내의 일정한 영역으로 운항하며 자신이 존재하는 좌표에서 관제소가 알려주는 위치로 이동한다.

④ 컨테이너를 하역중인 컨테이너 선박은 정박하며 일정 시간 이후 최종 좌표로 이동하게 된다.

시뮬레이션 프로그램은 성능의 측정을 위하여 기존의 DDM 서비스를 이용하는 방법과 선택적으로 Agent를 적용하는 방법 그리고 모든 데이터필터링에 Agent를 적용하는 방법으로 구현되었으며 이러한 성능을 측정하기 위하여 시뮬레이션에서 교환되는 메시지의 양을 측정하고 시뮬레이션 전체 수행 시간을 측정하는 코드를 작성하였으며 프로그램의 수행 모습은 그림 6와 같다.

〈표 2〉 구현 프로그램 파라미터

파라미터 항목	값
Routing Space	10km * 10km
관제소의 위치	관제소 1 (5km, 1km) 관제소 2 (5km, 10km)
레이더의 범위	5km * 9km
컨테이너 선박의 레이더 범위	2km * 2km
컨테이너 선박의 속도	2km/h
Simulation Time	100 step



〈그림 6〉 항만 시뮬레이션 프로그램

5. 결 론

Agent를 선택적으로 적용한 데이터필터링 기법은 기존의 DDM 서비스와 Agent만을 사용한 데이터필터링 기법의 문제점을 다음과 같이 해결하게 된다.

가. DDM 서비스의 문제점인 Update region과 Subscribe region의 변경 시 발생하는 비용을 줄일 수 있다.

나. 이동하는 시뮬레이션 객체에 대해 일대일로 연결하여 데이터를 교환함으로써 영역 변경시 발생할 수 있는 잘못된 데이터의 수신을 줄일 수 있다.

다. 이동하지 않는 시뮬레이션 객체는 기존의 DDM 서비스를 사용하므로 Agent를 사용함으로써 발생할 수 있는 메모리 공간의 낭비와 수행속도 저하를 최소화 할 수 있다.

본 논문에서 제시한 데이터필터링 기법을 사용한다면 워게임 같은 많은 수의 시뮬레이션 객체들이 참여하는 시뮬레이션과 객체들의 이동성이 큰 항공, 항만, 교통 시뮬레이션에서 보다 나은 성능을 보일 것으로 기대된다.

참고문헌

[1] U.S. Department of Defense, "High Level Architecture Interface Specification", Version 1.3, April 1998. http://www.dmso.mil

[2] Gray Tan and Liang Xu, "An Agentbased DDM for High Level Architecture", in Proceedings of Winter Simulation Conference. 1998

[3] Simon J.E. Taylor, "Modular HLA RTI Services : The GRIDS Approach", Proceedings of 6th International Workshop on Distributed Simulation and Real Time Applications