제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

HLA 기반 페더레이트 구조 개선을 위한 FOM 설계

서혜숙, 한상범, 신종회, 김태윤
고려대학교 컴퓨터학과
e-mail: suh@kida.re.kr, hansb@kt.co.kr, jshin@kisa.or.kr, tykim@netlab.korea.ac.kr

A FOM Design for Improved Federate Framework based on HLA

Heyi-Sook Suh, Sang-Beom Han, Jong-Whoi Shin, Tai-Yun Kim
Dept. of Computer Science, Korea University

요 약

오늘날 소프트웨어 개발은 재사용이 가능한 소프트웨어들을 컴포넌트로 가지는 프레임워크 개념을 실용화하는 추세로 발전하고 있다. 또한 재사용성과 이기종간의 상호 운용성을 보장하는 차세대 시뮬레이션 기술 구조인 HLA(High Level Architecture)에 기반을 두고 있는 많은 프레임워크들이 상용화되고 있다. 그러나 재사용성을 보장할 수 있는 컴포넌트를 개발한다는 것은 동적으로 페더레이트를 재구성할 수 있어야 한다는 새로운 도전이 기다리고 있다. 사실상 현재의 프레임워크들은 페더레이트가 만들어질 때 객체 모형의 컴포넌트들을 모두 알고 있어야 하는 정적인 객체 모형 표현 기법을 사용하고 있다.

본 연구에서는 RTI 를 사용하여 HLA 페더레이트를 구성하는 페더레이션 객체 모형(FOM)을 개선된 프레임워크 구조로 설계하였다. 제안된 프레임워크를 사용함으로써 개발 시간을 줄일 수 있는 것은 물론 개발자들은 시뮬레이션 관점에서 개발을 진행할 수 있다. 또한 이를 운용한 결과로써 유연성(Flexibility)을 크게 향상시킬 수 있었다.

1. 서론

대규모의 시뮬레이션 운영을 위해 다양한 컴포넌트(component)들을 통합하는 기술인 HLA(High Level Architecture)에서는 시뮬레이션 간의 상호 운용성을 증진시키기 위해 HLA 객체 모형 템플릿(OMT: Object Model Template)을 사용하여 객체 모형을 정의한다. HLA 프레임워크에 따라 구현된 소프트웨어의 최대 강점은 소프트웨어의 재사용성과 상호 운용성이 보장된다는 것이다. 이는 프레임워크 설계가 각 객체 모형에 독립적으로 이루어짐을 의미한다.

하나의 페더레이트(Federate)는 다음과 같은 두가지 방법으로 객체 모형을 표현하게 된다. 하나는 페더레이트가 만들어질 때(컴파일 시간) 객체 컴포넌트에 대해 모두 알고 있어야만 하는 정적인 객체 모형 표현과 다른 하나는 실행 시간에 결정되는 동적인 객체 모형 표현이 있다.

본 연구에서는 실행 시간에 시뮬레이션 객체 모형(SOM: Simulation Object Model)과 페더레이션 객체 모형(FOM: Federation Object Model)으로 바꾸는 새로운 객체 지향 프레임워크를 다루었으며, 페더레이트를 동적으로 바꿀 수 있도록 메타 표현 개념을 사용하였다.

제 2 장에서는 관련연구로써 객체 모형 템플릿(OMT)와 객체 지향 프레임워크를, 제 3 장에서는 페더레이트를 동적으로 바꿀 수 있는 메타 표현법을 살펴보았다. 제 4 장에서는 본 연구에서 제안하는 HLA 프레임워크의 FOM 설계 방법을 다루었다.

2. 관련 연구

2.1 객체 모형 템플릿(OMT)

HLA 객체 모형에 대한 정보를 표현하기 위한 공통적인 방법을 제공하는 것이 객체 모형 템플릿이다. 객체 모형 템플릿에 의해 정의되는 객체 모형에는 다음과 같은 세가지 유형이 있다.

- 객체 모형 유형
  1) 페더레이션 객체 모형(FOM)
     페더레이션에 의해 지원되는 필수적인 객체들

과 그 속성 및 상호 작용 식별자로 구성된다.

2) 시뮬레이션 객체 모형(SOM)
각각의 시뮬레이션이 페더레이션에 제공하는 고유한 능력들이 명시한다.

3) 관리 객체 모형(MOM: Management Object Model)
페더레이션 관리에 사용되는 객체와 상호 작용을 정의한다.

2.2 객체 지향 프레임워크

설계는 물론 코드까지 모두 재사용 가능한 소프트웨어 구조를 갖는 객체 지향 프레임워크는 일부분만 설계하거나 또는 문제된 영역만을 다시 구현하는 것이 가능하다.

블랙 박스 프레임워크

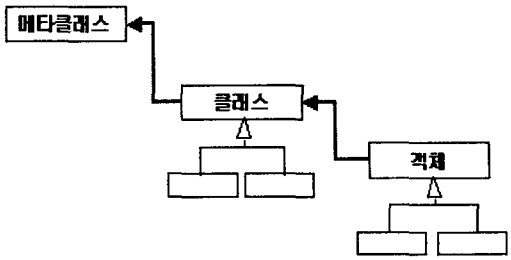
프레임워크를 개발하는 방법에는 화이트 박스 방법과 블랙 박스 방법이 있다. 상속에 의해서만 확장될 수 있는 프레임워크를 화이트 박스 프레임워크라 하며, 가지고 있는 컴포넌트들을 통해 확장되는 프레임 워크를 블랙 박스 프레임워크라 한다. 본 연구에서는 블랙 박스 프레임워크 방법을 사용한다.

메타 모형과 프레임워크

메타모형은 하나의 어플리케이션이나 처리과정, 또는 관련 객체의 그룹과 같은 구조를 설명할 수 있는 모형이다. 하나의 객체 메타모형은 객체 유형, 속성, 속성들의 유형, 객체들간의 관계 등을 정의할 수 있다. 정보를 가진 프레임워크는 코드를 추가할 것인지 아니면 수정할 것인지를 결정할 수 있다. 메타모형을 사용함으로써 데이터를 SOM 과 FOM 파일로 옮기고 코드를 다시 컴파일하지 않고 메모리에 객체를 다시 구축할 수 있다.

3. 메타 표현법

코드에 수정을 가하지 않고 다수의 FOM 들을 지원하는 프레임워크를 개발하는 한가지 방법은 <그림 1>과 같은 메타 클래스 개념을 사용하는 것이다. 만일 하나의 클래스가 객체로 표현된다면 그것을 다음 계층 단계에서는 클래스와 인스턴스로 만들 수 있다. HLA 는 개체를 클래스와 인스턴스로 정의할 수 있는 2 단계 시스템으로서 객체 모형 템플릿(OMT: Object Model Template)을 이용하여 객체와 클래스의 상호작용 구조를 정의함으로써 3 단계 시스템을 구축할 수 있다.

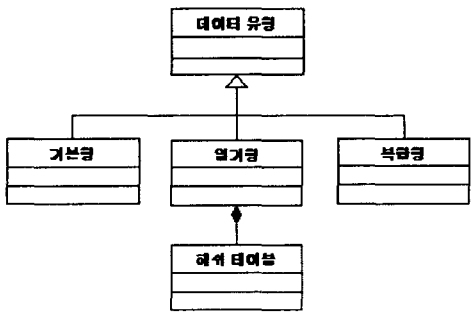


<그림 1> 3단계의 시스템 표현

메타클래스는 인스턴스와 클래스를 가지는 엔터티들이다. 클래스는 메타클래스의 인스턴스이다. 클래스들은 좀더 복잡한 유형의 속성을 가지며 일반화하기가 어렵다. OMT 는 유일한 이름을 갖는 속성들의 데이터 집합에 대한 객체 클래스를 정의한다. 객체 클래스는 상속 트리 구조를 가지며, 메타모형을 만들기 위해 사용되어진다. 객체 클래스 모형은 참조(subscribe) 또는 공표(publishable)와 같은 페더레이트에 대한 유용한 정보를 제공한다.

속성 및 파라메터 모형은 시뮬레이션과 페더레이션 객체에 내포되어 있는 속성과 상호작용 파라메터들을 자세히 기술해 준다. 속성 메타 데이터는 이름과 소유권, 전송 및 배달 순서, 갱신 권한 등과 같은 특성들의 집합으로 이루어진다. datatype 은 값의 유형과 OMT 속성 및 파라메터를 표현하기 위한 메타데이터이다.

프레임워크 내에 표현되어 질 데이터 유형은 <그림 2>와 같다. 기본형(base type)은 integer, byte, double, float, long & short 등 자바에서 제공하는 기본적인 유형과 스트링을 표현하는 클래스들을 만든다. 열거형(enumerated type)은 값에 관련된 이름을 해쉬 테이블을 이용하여 만든다. 복합형(complex type)은 전체를 계층적으로 표현하기 위해 객체들을 트리 구조로 만든다.

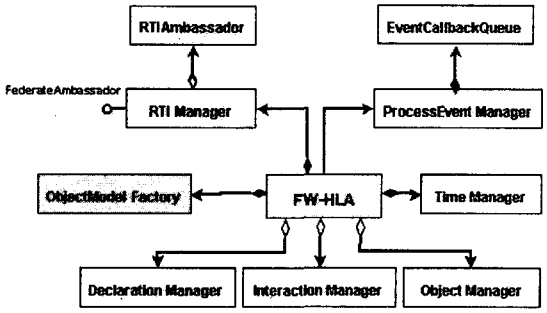


<그림 2> 메타데이터 표현 예

4. 프레임워크 구조 설계

프레임워크는 페더레이트와 자바 RTI 간의 한 계층을 제공하기 위해 자바 클래스들의 집합으로

설계하였다(<그림 3> 참조).



<그림 3> 프레임워크 구조 요약

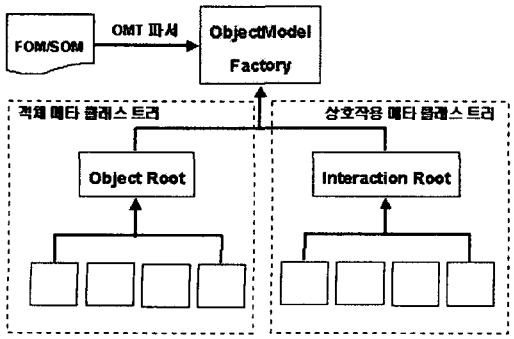
4.1 FW-HLA

프레임워크 구조를 갖는 HLA 를 FW-HLA 라 하며 프레임워크의 시작점을 의미한다. FW-HLA 는 페더레이트를 초기화하기 위해 SOM/FOM 파일 위치, 페더레이션 이름, 페더레이트 이름, 시간 관리 파라메터 등으로 구성된 파일을 읽어온다. FW-HLA 가 제공하는 메소드들은 다음과 같다.

- 페더레이션 생성 또는 연합 및 탈퇴
- 객체 속성 및 상호작용(interaction) 공표 또는 참조
- 사건의 발생 시간을 알리는 ReceiveInteraction, RemoveObject 와 같은 RTI 콜백 함수 등록
- 상호작용 발송, 객체 인스턴스 생성 및 삭제, 속성 갱신
- time constrained/regulate, loockahead, time step 등의 시간 관리 파라메터 수정

4.2 ObjectModel Factory

ObjectModel Factory 는 SOM 과 FOM 파일을 읽기 위한 파서로 사용된다. 객체와 상호작용을 갖는 클래스 정의에 사용되는 것이 메타 클래스이다. 객체 트리의 루트를 ObjectRoot 라 하고 상호작용 트리의 루트를 InterctionRoot 라 한다(그림 4> 참조).



<그림 4> ObjectModel Factory

Object Manager 와 Interaction Manager 가 객체를 생성하고자 원할 때 ObjectModel Factory 는 대응되는 트리에서 특정 이름의 메타 클래스를 찾아 이것을 복사한 후 요청자에게 되돌려준다.

4.3 RTI Manager

RTI Manager 는 FedarateAmbassador 인터페이스를 구현하고 객체들에게 이벤트 메커니즘을 제공한다. 예를 들어 DiscoverObject 라는 콜백 함수가 나타나면 RTI Manager 는 필요한 정보까지 포함해서 DiscoverObjectEvent 를 생성하여 등록된 참여자에게 전달한다.

4.4 Declaration Manager

Declaration Manager 는 클래스와 상호작용의 속성 값을 공표하거나 참조하도록 하는 선언 기능을 수행한다. RTI 에게 선언하는 메타 클래스 트리에 포함된 정보를 사용하여 자동적으로 생성된다.

4.5 Object Manager

객체 플록시는 분산 객체들을 관리하기 위해 사용되는 메커니즘으로써 근거리 및 원거리의 플록시는 Object Manager 에 의해 관리된다. HLA 객체 인스턴스가 페더레이션이나 프렘임워크에 의해 등록되면 Object Manager 는 인스턴스 유형에 관련된 메타 클래스를 가진 플록시를 생성하여 삽입시킨다. 일단 플록시가 생성되면 Object Manager 는 근거리나 원거리에 관계없이 자동적으로 객체 속성을 갱신한다.

4.6 Interaction Manager

프레임워크는 RTI 에게 상호작용을 보내기 위해 Interaction Manager 를 사용한다. 페더레이션에 참가한 참여자 간의 상호작용이 발생하면 Interaction Manager 는 모든 등록된 참여자에게 이 사실을 통보한다. Interaction Manager 는 항상 페더레이트가 참조할 수 있도록 송신한 상호작용들을 큐에 유지해야만 한다.

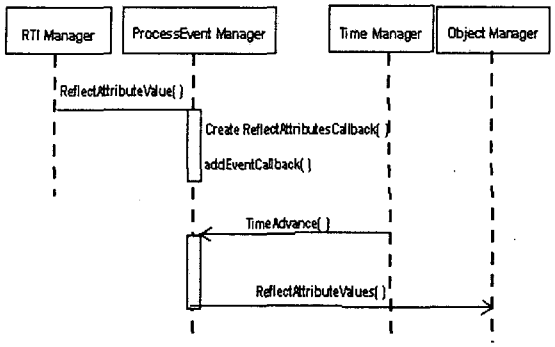
4.7 Time Manager

Time Manager 는 페더레이션에 있는 페더레이트 간의 시간 진행을 동기화시킨다. 페더레이트는 자신의 시간 진행을 요청한 후 RTI 로부터 그 진행을 보장하는 콜백 함수가 도착할 때까지 기다린다. Time Manager 는 현재 Time Stepped Simulation 과 Event-Driven Simulation 의 2 가지 유형을 사용하고 있다.

4.8 Process Event Manager

Process Event Manager 는 프레임워크에 있는 모든 사건들과 수정사항 등을 알려준다. 예를 들어 Process Event Manager 가 Time Manager 로부터 신호를 받으면 시간 순서에 맞는 이벤트 콜백 함수를 처리

한다(<그림 5> 참조).



<그림 5> Process Event 예

5. 결론

본 연구에서 다룬 HLA 프레임워크 설계는 객체 기반의 메타 표현법을 사용하여 실행시 객체를 동적으로 할당함으로써 코드의 수정없이 서로 다른 SOM 혹은 FOM 을 사용하는 페더레이션에도 간단히 참여할 수 있는 유연성을 제공하고 있다.

그러나 이러한 프레임워크를 사용함에 있어 사용자들이 고려해야 할 것은 유연성이 보장되는 반면 성능면에서는 그렇지 않을 수도 있다는 점이다. 실험한 결과 동적인 표현법을 사용할 경우 페더레이트가 관리할 객체의 수가 많아지면 성능이 감소되는 것으로 나타났다. 특히 이벤트 메커니즘을 사용할 경우 메시지 교환량에 따라 병목현상을 일으키기도 하였다. 그러므로 본 연구에서 제안한 프레임워크를 사용하는 문제는 개발자의 선택의 문제이며, 유연성과 성능을 모두 보장할 수 있는 새로운 연구가 계속되어야 할 것이다.

참고문헌

1. 한국국방연구원, 국방모의분석체계 구축을 위한 상위체계구조(HLA) 기술 연구, 연구보고서, 1999.
2. DMSO, DoD HLA Run-Time Infrastructure Programmer's guide RTI 1.3 Ver.6, 1999, URL=http://www.dmso.mil.
3. DMSO, DoD HLA Interface Specification Version 1.3, 1999, URL=http://www.dmso.mil.
4. DMSO, HLA/RTI Verification, 2000, URL=http://www.dmso.mil.
5. Erich Gamma et al, Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
6. Frederick Kuhl, Richard Weatherly, Judith Dahmann, Creating Computer Simulation Systems, Prentice Hall PTR, 2000.
7. Kevin Cox, A Framework-based Approach to HLA Federate Development, .Fall Simulation Interoperability Workshop, 1998.
8. Proposed IEEE Standard P1516.1, Standard for Modeling and Simulation High Level Architecture □ Object Model Template Specification, draft 5, 2000.
9. Rassul Ayani, Gary Tan, Farshad Moradi, Liang Xu, Yusong Zhang, Distributed Real-time Simulation and High Level Architecture, 1999, URL=http://www.comp.nus.edu.sg/~rpsim
10. Regis Dumond, A FOM Flexible Federate Framework, Spring Simulation Interoperability Workshop, 2001,.
11. Roy Crosbie, John Zenor, High Level Architecture Training Module 1-6, California State University, 2000.