한국시뮬레이션학회 논문지
Vol. 17, No. 1, pp. 43-52 (2008. 3)

IEEE 1516 HLA/RTI 표준을 만족하는 시간 관리 서비스 모듈의 설계 및 구현

홍정희[1†] · 안정현[1] · 김탁곤[1]

Design and Implementation of Time Management Module for IEEE 1516 HLA/RTI

Jeong Hee Hong · Jung hyun Ahn · Tag Gon Kim

ABSTRACT

The High Level Architecture (HLA) is the IEEE 1516 standard for interoperation between heterogeneous simulators which are developed with different languages and platforms. Run-Time Infrastructure (RTI) is a software which implements the HLA Interface Specification. With the development of time management service of RTI, it is necessary to consider an efficient design approach and an algorithm of Greatest Available Logical Time (GALT) computation. However, many time management services of existing RTIs have difficulty in modification and extension. Although some RTIs avoid this difficulty through modular design, they comply with not IEEE 1516 HLA/RTI but HLA 1.3. In addition, a lot of RTIs made use of well-known Mattern's algorithm for GALT computation. However, Mattern's algorithm has a few limitations for applying to IEEE 1516 HLA/RTI.

This paper proposes a modular design and an implementation of time management service for IEEE 1516 HLA/RTI. We divided th time management service module into two sub-modules: a TIME module and a GALT module and used Mattern's algorithm improved for IEEE 1516 HLA/RTI. The paper also contains several experimental results in order to evaluate our time management service module.

Key words : IEEE 1516, HLA, RTI, Time management, Modular design, GALT calculation algorithm

요 약

HLA는 이기종 시뮬레이터간의 연동을 위한 상위레벨 개념으로서 IEEE 1516 표준이며 Run-Time Infrastructure (RTI)는 이를 구현한 라이브러리 형태의 소프트웨어이다. RTI의 시간 관리 서비스를 개발함에 있어 효율적인 설계와 GALT 계산 알고리즘은 반드시 고려해야 하는 사항이다. 그러나 기존의 많은 RTI의 시간 관리 서비스는 수정과 확장이 용이하지 않다. 몇몇 RTI가 이러한 단점을 극복하기 위하여 모듈화 설계방안을 제안하고 있지만 이들은 IEEE 1516 HLA/RTI에 적합하도록 설계 및 구현된 것이 아니라 HLA 1.3을 위한 것이었다. 많은 RTI는 GALT 계산 알고리즘으로 분산 시뮬레이션에서 널리 알려진 Mattern의 알고리즘을 이용하고 있다. 그러나 Mattern의 알고리즘을 IEEE 1516 HLA/RTI에 적용시키기에는 몇 가지 제약사항이 존재한다.

본 논문은 IEEE 1516 HLA/RTI를 만족하는 시간 관리 서비스의 모듈화 설계를 제안하고 이를 바탕으로 구현한다. 시간 관리 서비스 모듈을 TIME 모듈과 GALT 모듈로 나누는 설계방안을 제안하고 GALT 계산 알고리즘으로는 Mattern의 알고리즘을 수정·보완하였다. 그리고 제안한 설계방안대로 구현한 시간 관리 서비스 모듈의 성능을 살펴보기 위하여 실험을 수행하고 그 결과를 분석하였다.

주요어 : IEEE 1516, HLA, RTI, 시간 관리 서비스, 모듈화 설계, GALT 계산 알고리즘

* 본 연구는 과학재단 특정기초연구(R01-2006-000-111180) 지원으로 수행되었다.
2008년 3월 12일 접수, 2008년 3월 24일 채택
[1] 한국과학기술원 전자전산학과
주 저 자 : 홍정희
교신저자 : 홍정희
E-mail; jhhong@smslab.kaist.ac.kr

1. 서 론

High Level Architecture(HLA)는 이기종 시뮬레이터 간의 연동을 위한 IEEE 1516 표준이며 Run-Time Infrastructure(RTI)는 이를 구현한 라이브러리 형태의 소프트웨어이다[1-3].

RTI에서 시간 관리 서비스는 시뮬레이터 사이에서 시간 동기화를 위해 중요한 부분이다. 시간 관리 서비스는 페더레이트의 정책을 결정하고 각 페더레이트의 시간 진행을 관리한다. 페더레이트의 정책은 레귤레이팅(Regulating)과 컨스트레인드(Constrained)의 조합으로 이루어질 수 있다. 레귤레이팅 페더레이트는 Time-Stamp Order(TSO) 메시지를 보낼 수 있으며, 컨스트레인드 페더레이트는 TSO 메시지를 시간이 감소하지 않는 순서로 수신한다. 시간 인과성(Time Causality)를 유지하기 위하여 컨스트레인드 페더레이트는 Greatest Available Logical Time (GALT)보다 큰 시간으로는 시간 진행을 할 수 없다. 즉, GALT는 페더레이트가 안전하게 진행할 수 있는 최대 시간을 의미한다.

HLA는 구현물인 RTI에 대해 특정한 프로그래밍 언어나 구조를 명시하고 있지 않으며 현재 많은 RTI가 배포되어 있다. 그러나 각각의 RTI는 그 설계 구조나 구현에 대해서 공개하지는 않고 있다. 몇몇의 RTI는 그 내부를 공개하고 있으나 시간 관리 서비스가 RTI에 임베디드되어 있음으로 인해 시간 관리 서비스만을 수정하거나 확장하기에는 어려운 구조를 취하고 있다. 이러한 결점을 피하기 위해 본 논문에서는 모듈화된 설계를 취하고 있다. 모듈화는 수정·유지보수의 용이성, 재사용성을 높여준다. 현재 알려진 바로는 IEEE 1516 표준에 맞는 RTI를 이와 같은 방법으로 설계한 예는 없다. 본 논문은 이와 같이 개발한 RTI를 SMSRTI[4]라 부르며 본 논문에서는 SMSRTI의 한 부분인 시간 관리 서비스 모듈의 설계와 구현에 대해서 다루고자 한다.

시간 관리 서비스 모듈을 설계 및 구현시 고려해야 하는 두 가지 사항은 GALT 계산 알고리즘과 효율적인 설계 방안이다.

먼저, GALT 계산 알고리즘은 시간 관리 서비스의 성능에 상당한 영향을 미치는 요소로서 본 논문에서는 많은 RTI가 기반으로 사용하고 있는 Mattern의 알고리즘[5]을 이용한다. Mattern의 알고리즘이 분산 시뮬레이션에서 널리 알려진 알고리즘이기는 하나 IEEE 1516 HLA/RTI에 적용함에 있어 몇 가지 제약사항이 존재한다. 본 논문에서는 이러한 제약사항을 극복하기 위하여 Mattern의 알고리즘을 수정·보완하였다.

다음으로 시간 관리 서비스 모듈의 효율적인 설계를 위하여 시간 관리 서비스 모듈을 TIME 모듈과 GALT 모듈로 나눔으로써 모듈화된 설계 프레임워크를 구축하였다. 시간 관리 서비스 모듈내의 두 개의 모듈과 SMSRTI 내의 다른 서비스 모듈과의 상호작용을 위하여 인터페이스를 정의하고 두 모듈사이의 인터페이스도 정의하여 재사용성을 높이고 수정을 용이하게 하였다. 앞서 언급한 바와 같이 GALT 계산 알고리즘은 시간 관리 서비스의 성능에 있어 매우 중요한 부분이다. 이러한 여러 GALT 계산 알고리즘을 적용하고 비교하고자 할 때, 우리의 시간 관리 서비스 모듈은 GALT 모듈만을 교체함으로써 그 수고를 덜어준다. 이는 시간 관리 서비스 모듈의 두 모듈이 서로 독립적으로 동작하며 TIME 모듈은 완벽히 재사용될 수 있기 때문이다.

본 논문의 구성은 다음과 같다. 2절에서는 HLA에 대해서 간략히 살펴보고자 한다. 3절에서는 Mattern의 알고리즘에 대해 살펴보고 본 논문에서 수정한 알고리즘에 대해서 기술한다. 그리고 4절에서는 시간 관리 서비스 모듈의 설계에 대해 언급하며 5절에서는 개발된 시간 관리 서비스 모듈의 정확도와 성능을 측정하기 위해 수행한 실험에 대해서 살펴본 뒤 결론을 맺는다.

2. High Level Architecture

2.1 High Level Architecture(HLA)

미 국방성에서 제안된 HLA는 이기종 시뮬레이터간의 연동을 위한 상위레벨 개념으로서 2000년도에 IEEE 1516 표준으로 채택되었다. HLA는 HLA 프레임워크와 규칙(HLA Framework and Rules)[1], 페더레이트 인터페이스 명세(Federate Interface Specification)[2] 그리고 객체 모델 템플릭(Object Model Template - OMT)[3]의 세 가지로 정의된다.

먼저, HLA 프레임워크와 규칙은 페더레이션(Federation)에 포함되는 구성 요소들의 역할과 상호 관계에 관한 전반적이고 기본적인 10개의 규칙들이다. 페더레이트 인터페이스 명세는 각 페더레이트(Federate)와 RTI 간의 기능적 인터페이스에 관한 규약으로 6가지의 서비스 관리 영역으로 나누어 기술하고 있다. RTI는 페더레이트 인터페이스 명세를 시스템 기종 및 프로그램 언어별로 라이브러리 형태로 구현한 것이다. 마지막으로 OMT는 페더레이션을 구성하는 페더레이트들 간에 이루어지는 공통 데이터 영역을 구조적, 기능적으로 서술하는데 사용된다. 페더

레이션을 구성하는 페더레이트들 사이의 공유 데이터 교환 구조를 서술하는 FOM(Federation Object Model)과 시뮬레이션 시스템이 페더레이션에 제공하는 기능을 표현하는 SOM(Simulation Object Model), 그리고 MOM (Management Object Model)으로 구성된다.

2.2 Time Management Service

분산 시뮬레이션에서 시간 관리 서비스의 가장 큰 목적은 각 페더레이트의 시간 진행이나 메시지의 송수신에 있어서 인과 관계(Causality)를 보장해주는 것이라고 할 수 있다. 페더레이션 내에서 시간은 항상 증가하는 방향으로 진행되어야 하는데, 페더레이션에 참여하고 있는 페더레이트의 현재 시간은 서로 다르기 때문에 페더레이션 시간 축에 있는 각 페더레이트의 시간 진행을 조정해야 할 필요가 있다. 이를 통해 페더레이트들에게 전달되는 정보는 원인과 결과가 정확하고 순차적이어야 한다[6]. 즉, 시간 관리 서비스는 페더레이션 내의 논리 시간 진행과 Time-Stamp 데이터가 전달되는 순서를 조정한다. 이러한 시간 관리 서비스에서 정의되는 메시지의 유형, 시간 관리 정책, 시간 진행 방법 그리고 시간 진행과 메시지 처리에 있어 중요한 변수가 되는 GALT의 계산에 대해서 살펴본다.

페더레이트가 수신하는 메시지는 RO(Receive Order) 메시지와 TSO 메시지로 나눠질 수 있다. 전송한 순서와 메시지의 시간 정보 유무에 관계없이 페더레이트에 수신된 순서대로 전달되어 처리되는 메시지를 RO 메시지라고 하며 이 메시지를 처리하는 것은 논리 시간의 흐름에 의해 영향을 받지 않는다. RO 메시지와는 달리 시간적 의미를 갖는 메시지로 수신한 순서에 관계없이 Time-Stamp 순서대로 처리해야하는 메시지를 TSO 메시지라고 하며 이 메시지는 페더레이트의 논리 시간의 흐름에 영향을 받는다.

위에서 설명한 메시지를 전송하고 수신하는 것은 페더레이트가 어떠한 정책을 가지고 있느냐에 달려있다. 레귤레이팅 페더레이트는 컨스트레인드 페더레이트의 시간 진행을 제어한다. 일반적으로 페더레이트는 "Regulating", "Constrained", "Regulating and Constrained", "Non-Regulating and Non-Constrained"의 4가지 중에서 하나의 정책을 택할 수 있다. 레귤레이팅 정책을 택한 페더레이트는 TSO 메시지를 생성할 수 있다. 이 때, 생성되는 TSO 메시지의 Time-Stamp는 Lookahead 라는 값에 영향을 받는다. TSO 메시지의 Time-Stamp는 "현재시간 + Lookahead"보다 이전 시간이어서는 안 된다. 즉, "현재시간 + Lookahead"보다 작은 TSO 메시지는 발생시키지 않는다는 것이다. Lookahead 값은 페더레이트가 레귤레이팅을 선택하는 시점에서 반드시 정의해야 하나 시뮬레이션 도중에 변경시킬 수 있다. 컨스트레인드 정책을 택한 페더레이트는 TSO 메시지를 수신할 수 있다. 컨스트레인드 정책을 취하지 않는 페더레이트도 다른 레귤레이팅 페더레이트가 보낸 TSO 메시지를 받기는 하지만 TSO 메시지를 마치 RO 메시지인 것처럼 처리한다. 레귤레이팅 페더레이트가 Lookahead라는 정보를 가지고 있는 것처럼 컨스트레인드 페더레이트는 시간 진행을 위한 값으로 GALT라는 정보를 가지고 있다. 여기서 GALT란, 페더레이트가 받을 수 있는 TSO 메시지 중에서 가장 최근의 시간 값이다. 즉, 자신을 제외한 모든 레귤레이팅 페더레이트들의 "현재시간 + Lookahead" 값 중 최소값이 되는 것이다. 컨스트레인드 페더레이트는 GALT 이후로는 시간 진행을 할 수 없다. 왜냐하면 GALT 이후의 시간까지 진행해버리고 난 후에 그 시간보다 작은 TSO 메시지가 오지 않는다는 것을 보장할 수 없기 때문이다. 즉, GALT는 페더레이트에게 GALT값보다 Time-Stamp가 작은 TSO 메시지는 이 후에는 오지 않는다는 것을 보장해주는 것이다. 그림 1은 페더레이트 정책에 관한 예를 보여주고 있다.

3. GALT 계산 알고리즘

GALT는 페더레이트가 시간 인과성을 깨지 않고 안전하게 진행할 수 있는 최대 시간을 의미한다. 수식으로 표현하면 다음과 같다.

$$GALT = \min\{\text{페더레이트 } F_i\text{의 현재 시간} + \text{페더레이트 } F_i\text{의 Lookahead}\}$$

GALT는 HLA 1.3[7]에서 정의하고 있는 Lower Bound

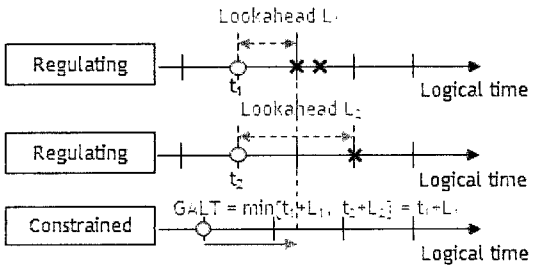


그림 1. 페더레이트 정책

Time Stamp(LBTS)와 일맥상통하며 GALT 계산은 시간 관리 서비스에 있어서는 필수적인 요소이다.

GALT를 계산함에 있어서 가장 중요한 것은 과도 메시지(Transient message) 문제를 해결하는 것이다. 과도 메시지란 보내는 페더레이트에서 메시지를 보냈으나 수신하는 쪽에서 네트워크상의 시간 지연 등의 이유로 아직 수신하지 못한 메시지이다[8]. 그러나 이러한 메시지는 시간 지연은 있을지라도 정확하게 도착하는 메시지이므로 GALT를 계산함에 있어서 과도 메시지 문제를 고려하지 않는다면 시간 인과성을 위배하는 메시지가 도착할 수도 있다. 따라서 GALT 계산 알고리즘은 보내고 받는 모든 메시지를 고려해야만 한다.

현재까지 과도 메시지 문제를 해결하기 위한 많은 알고리즘이 제안되어 왔다. Samadi가 제안한 알고리즘[9]은 메시지를 받은 프로세스가 메시지를 보낸 프로세스에 승인(Acknowledgement) 메시지를 보내도록 함으로써 해결하였으나 Unreliable 채널과 non-FIFO 채널에 적용될 수 있도록 제안되었기 때문에 모든 메시지에 대해 쌍을 이루는 승인 메시지가 필요하므로 네트워크상에서 상당한 트래픽을 유발할 수 있다. Lin과 Lazowska의 알고리즘[10]은 Samadi의 알고리즘을 개선한 것으로 모든 메시지에 대해 승인 메시지를 보내는 것 대신에 링크드 리스트 데이터 구조를 이용하는 방법을 제안했으나 각 프로세스는 프로세스들 사이에서 주고받는 메시지 쌍을 위해 같은 데이터 구조로 유지해야하므로 각 프로세스의 Fan-in/Fan-out의 수가 많아지면 상당한 양의 메모리가 필요할 뿐만 아니라 처리하는데 걸리는 부하도 늘어나게 된다. 그리고 가장 널리 사용되고 있는 Mattern의 알고리즘[5]은 Coloring Scheme과 벡터 카운터 방법을 이용하여 주고받은 메시지를 세도록 하여 해결하였다.

본 논문에서 사용하고 있는 알고리즘은 분산 시뮬레이션에서 가장 많이 사용되고 있는 Mattern의 알고리즘에 기반하고 있다. 그러나 Mattern의 알고리즘을 HLA의 시간 관리 서비스를 위하여 바로 적용하기에는 부합되지 않는 사항이 존재하므로 이를 개선하고 추가적인 고려사항 또한 고려하여 HLA/RTI에 적합하도록 Mattern의 알고리즘을 수정하였다.

3.1 Mattern의 알고리즘

Mattern의 알고리즘은 과도 메시지 문제를 해결하기 위해 가장 널리 알려진 알고리즘 중의 하나로 Coloring Scheme과 벡터 카운터 방법을 제안하였다. Mattern은 설명의 편의를 위해 각 프로세스가 링 토폴로지를 이루고 있다고 가정하고 있다.

Coloring Scheme은 각각의 프로세스는 색깔을 띠고 있으며 프로세스가 보내는 메시지는 메시지가 보내질 때의 프로세스의 색깔을 따른다고 가정하고 있다. 즉, 빨간색의 프로세스가 보내는 메시지는 빨간색 메시지라는 것이다. 프로세스가 다른 프로세스로부터 컨트롤 메시지를 받을 때, 그 지점을 컷 지점(Cut Point)이라고 하며 그 점을 기점으로 프로세스의 색깔을 바꾼다. 각 프로세스의 컷 지점을 이은 것을 컷(Cut)이라고 한다.

벡터 카운터 방법의 기본 개념은 컷을 통과하는 메시지, 즉 과도 메시지를 센다는 것이다. 이는 Coloring Scheme에 기반하여 각 프로세스가 주고받는 메시지를 센다는 것을 의미한다. 이를 이용하여 과도 메시지 문제를 해결할 때 최대한 2개의 컷만 있으면 된다. 첫 번째 컷 이전의 프로세스의 색깔을 빨간색이라고 하고 첫 번째 컷 이후에 프로세스가 초록색으로 바뀐다고 하자. 이 때, GALT 계산을 위해 고려해야 하는 과도 메시지는 빨간색 메시지가 된다. 위에서 설명한 바와 같이 메시지의 색깔은 보내는 프로세스의 색깔을 따르므로 빨간색 메시지는 첫 번째 컷 이 전에 보내진 메시지이다. 따라서 첫 번째 컷 이후에는 빨간색 메시지를 발생시킬 수 없으므로 첫 번째 컷에서 각 프로세스로 보내진 빨간색 메시지가 몇 개인가의 정보를 얻을 수 있다.

벡터 카운터 방법에서 모든 각각의 프로세스 P_i는 자신이 보내고 받은 빨간색 메시지를 세는 벡터 카운터 V_i를 가지고 있다. 이 때, 벡터 카운터 V_i의 길이는 총 프로세스의 개수가 된다. 프로세스 P_i가 프로세스 $P_j(i \neq j)$로 빨간색 메시지를 보낼 때마다 벡터 카운터 $V_i[j]$를 증가시킨다. 즉, $V_i[j] := V_i[j]+1$ 이 된다. 그리고 프로세스 P_i가 프로세스 $P_j(i \neq j)$로부터 빨간색 메시지를 받을 때는 벡터 카운터 $V_i[j]$를 감소시킨다. 즉, $V_i[j] := V_i[j]-1$ 이 된다. 컨트롤 메시지가 가지고 있는 카운터 벡터 C는 컨트롤 메시지가 링을 따라 전달되면서 각 프로세스의 벡터 카운터 V와 더한 값을 컨트롤 메시지의 카운터 벡터 C에 저장하여 다음 프로세스로 전달한다. 즉, 프로세스 P_i가 컨트롤 메시지를 받았을 때, $C := C+V_i$ 가 된다. 프로세스 P_i가 다음 프로세스로 컨트롤 메시지를 전달하기 위해서는 $C[i]+V_i[i] \leq 0$ 이라는 조건을 만족해야 한다.

링을 따라 전달되는 컨트롤 메시지가 한 바퀴를 다 돌아서 처음 컨트롤 메시지를 보냈던 프로세스로 돌아온 것을 라운드라고 한다. 첫 번째 라운드가 끝난 후, 컨트롤 메시지의 카운터 벡터 $C[i]$는 프로세스 P_i의 과도 메시지의 수를 의미한다. 만약 첫 번째 라운드에서 컨트롤 메시

지의 카운터 벡터 C[i]가 0보다 크다면, 두 번째 라운드, 즉 두 번째 컷을 필요로 하게 된다.

두 번째 라운드에서 컨트롤 메시지를 받은 프로세스 P_i는 다음 프로세스로 컨트롤 메시지를 전달하기 위한 조건 $C[i]+V_i[i] \leq 0$을 만족시키기 위해, 모든 빨간색 메시지가 도착할 때까지 기다려야만 한다. 그러므로 두 번째 라운드가 끝나고 나면 컨트롤 메시지의 벡터 카운터 C 는 필연적으로 0이 되어 과도 메시지를 모두 받은 상태가 된다. 이 때, GALT 값을 계산할 상태가 되었음을 의미한다. 그림 2는 위에서 설명한 Coloring Sceme과 벡터 카운터 방법을 이용하여 메시지를 세는 Mattern의 알고리즘의 예를 보여주고 있다.

3.2 수정된 Mattern의 알고리즘

Mattern의 알고리즘이 널리 사용된다 할지라도 HLA/RTI에 적용시키기에는 몇 가지 제약사항이 존재한다. 본 논문에서는 그러한 제약사항을 극복하기 위해 Mattern의 알고리즘을 수정·보완하였다. 본 논문에서의 알고리즘이 기본적으로는 Mattern 알고리즘과 유사하지만, Reliable 채널뿐만 아니라 Best-effort 채널도 고려하고 있다는 점과 중앙식 구조를 취하고 있다는 점에서 크게 다르다.

모든 메시지는 Reliable 채널 즉, TCP/IP를 통해서만 전달되는 것이 아니라 Best-effort 채널 즉, UDP/IP를 통해서도 전달될 수 있다. 모든 Reliable 또는 Best-effort 메시지는 TSO 메시지가 될 수 있다. HLA/RTI의 이러한 사항은 Mattern의 알고리즘에서는 제약사항이 될 수 있다. 이는 Reliable 과 Best-effort의 두 가지 유형의 채널은 전송 도중 손실될 수 있느냐와 시간 인과성을 어길 수 있느냐에 대한 특성의 차이에서 비롯된다. Reliable 메시지는 절대 손실되지 않으며 시간이 감소하는 순서로 도착하지 않는다. 반면에 Best-effort 메시지는 전송 도중 손실될 수 있으며 Time-Stamp 순서의 관점에서 시간 순서대로 도착한다는 보장이 없다. 이러한 차이로 인한 Mattern 알고리즘의 제약사항은 그림 3과 같다. 실선으로 표현된 메시지는 Reliable 방식으로 전송되는 메시지이며 점선으로 표현된 메시지는 Best-effort 방식으로 전송되는 메시지라고 하자.

CASE 1: 전송 도중에 손실되는 메시지가 있다.(M_2)
CASE 2: 페더레이트의 현재 시간보다 작은 Time-Stamp를 가지는 메시지가 도착한다.(M_3)

만약 Best-effort 방식으로 전송되는 메시지도 Mattern의 알고리즘에서 사용하는 벡터 카운터 방법을 그대로 적용할 경우 CASE 1의 M_2처럼 전송 도중에 손실되는 경우에 과도 메시지 계산이 종료될 수 없다. 페더레이트 A가 페더레이트 B로부터 컨트롤 메시지를 받은 후, 다음 페더레이트 C로 컨트롤 메시지를 전달하기 위한 조건인 $C_A[A]+V_A[A]$가 M_2의 손실로 결코 0이 될 수가 없기 때문이다. 그러므로 페더레이트 A는 페더레이트 C로 컨트롤 메시지를 전달하지 못하고 계속 메시지 M_2를 기다리게 되는 것이다. 이렇게 되면 GALT 계산을 하지 못하게 되므로 페더레이트의 시간 진행 또한 할 수 없게 되어 데드락 상황이 발생하게 된다.

이를 해결하기 위해 본 논문에서는 Best-effort 방식으로 메시지를 전송할 때는 페더레이트에서 메시지 카운터 벡터 V에 그 정보를 반영하지 않도록 하였다. 즉, 메시지를 세지 않는다는 것이다. 반대로 메시지를 받았을 경우에도 마찬가지로 메시지를 세지 않는다. 이렇게 처리함으로써 CASE 1과 같은 상황으로 인한 데드락을 피할 수 있게 된다.

그러나 위와 같이 처리하였을 경우 CASE 2와 같은 문제가 발생할 수 있다. 이미 GALT를 계산한 이후에 도착한 Best-effort 방식으로 전송된 TSO 메시지의 Time-Stamp가 현재 시간보다 작은 값을 가지는 경우이다.

이를 해결하기 위해 본 논문에서는 메시지 M_3처럼 시

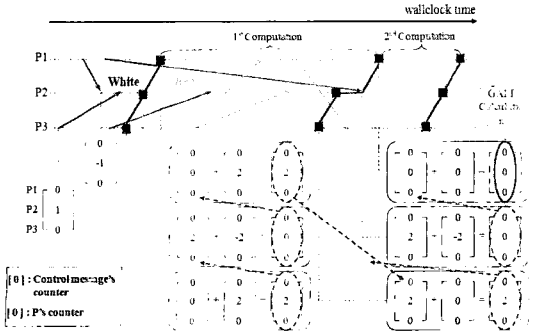


그림 2. Mattern의 알고리즘

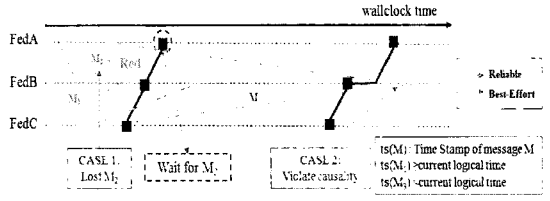


그림 3. Mattern의 알고리즘의 제약사항

간 인과성을 위배하는 메시지가 도달할 경우 메시지 큐에서 삭제시킴으로써 인과성을 보장하도록 한다. 즉, 메시지가 전송 도중에 손실된 것처럼 처리해주는 것이다. 이처럼 처리를 하더라도 무방한 이유는 바로 Best-effort 전송 방식의 특성 때문이다. 이는 전송 도중에 메시지가 손실될 수도 있다는 가정 때문이며, 만약 시뮬레이션에서 중요한 정보를 가지는 메시지라면 반드시 Reliable 방식으로 전송했을 것이라는 이유 때문이다.

위와 같이 Mattern의 알고리즘을 수정·보완하여 더 이상 제약사항이 존재하지 않음을 알 수 있다.

4. 시간 관리 서비스 모듈의 아키텍처

4.1 SMSRTI의 전체 아키텍처

SMSRTI는 그림 4에서 보듯이 DMSO RTI 1.3NG[7], pRTI[11], MÄK RTI[12], DRTI[13]와 마찬가지로 분산 아키텍처를 취하고 있으며, RTIExec(RTI Execution), FedExec(Fedeation Execution), 페더레이트로 이루어져 있다. RTIExec는 페더레이션의 생성/소멸을 관리하고 FedExec와 통신한다. FedExec는 개별 페더레이트가 페더레인션에 참가/탈퇴하는 것과 더불어 페더레이션에 참가한 페더레이트들의 정보를 관리한다. 각각의 FedExec는 다른 FedExec와는 독립적으로 동작한다. 페더레이트는 사용자 코드, Local RTI Component(LRC)로 이루어져 있으며, LRC는 RTI ambassador, 페더레이트 ambassador 그리고 각각의 서비스 모듈로 구성되어 있다. 사용자가 HLA Application Programming Interface(API)를 호출하면 RTI ambassador에서 호출된 서비스의 처리를 위해 LRC로 메시지를 보낸다. LRC는 필요할 경우 FedExec와 통신함으로써 서비스 처리를 하지만, 대부분의 경우는 LRC 내부에서 처리한다. 콜백이 되는 경우는 LRC가 페더레이트 ambassador로 그 메시지를 전달한다. FedExec와 개별 페더레이트의 LRC가 주고받는 메시지는 네트워크를 통해서 전달된다.

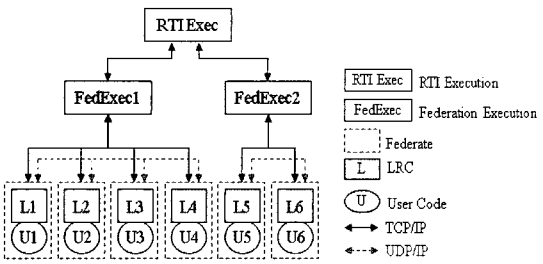


그림 4. SMSRTI의 전체 아키텍처

이 때, 메시지 패킷의 전송 방식은 Reliable 방식 또는 Best-effort 방식을 취한다. 이는 3절에서 언급한 대로 Reliable 방식은 TCP/IP와 같은 경우이고, Best-effort는 UDP/IP와 같은 경우이다.

4.2 시간 관리 모듈의 프레임워크

그림 5는 시간 관리 서비스 모듈의 프레임워크를 보여주고 있다. 굵은 실선으로 표시된 부분이 전체 SMSRTI 중에서 시간 관리 서비스를 나타낸다. 본 논문에서 제안하고 있는 시간 관리 서비스 모듈을 TIME 모듈과 GALT 모듈로 나뉘어진다.

먼저, TIME 모듈은 페더레이트가 정한 정책을 설정하고 시간 진행을 위한 API 서비스에 대한 처리를 담당하고 있다. IEEE 1516에서 페더레이트의 정책을 설정하는 API는 *Enable Time Regulation, Disable Time Regulation, Enable Time Constrained, Disable Time Constrained*가 있으며 시간 진행을 위해서는 *Time Advance Request, Time Advance Request Available, Next Message Request, Next Message Request Available, Flush Queue Request*가 있다.

다음으로, GALT 모듈은 GALT 계산을 주기적으로 수행하는 것으로 3절에서 설명한 수정된 Mattern의 알고리즘을 이용하고 있다. GALT 계산 알고리즘의 구현을 위하여 중앙식 구조를 택하였다. 중앙식 구조는 시뮬레이션 도중에 페더레이트가 참가하거나 탈퇴하는 경우와 Best-effort 방식으로 메시지를 전송하는 경우를 처리하기가 용이하기 때문이다.

두 모듈 사이의 메시지 교환을 위하여 인터페이스를 정의하고 시간 관리 서비스 모듈과 SMSRTI 내의 다른 모듈과의 상호작용을 위해서도 인터페이스를 정의하였다.

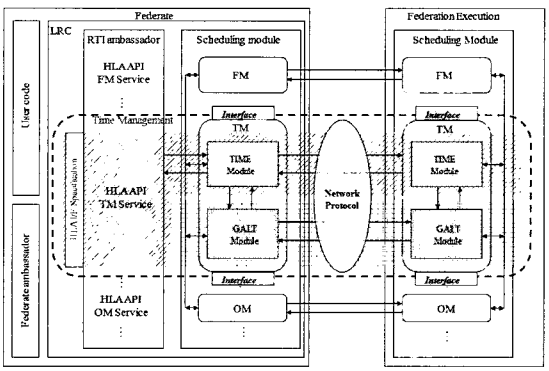


그림 5. 시간 관리 서비스 모듈의 프레임워크

페더레이트의 LRC와 FedExec가 주고받는 프로토콜도 정의하였다.

시간 관리 서비스 모듈의 앞서 설명한 두 개의 모듈로 나누어 설계 및 구현함으로써 재사용성을 높이고 수정 및 유지보수를 쉽게 할 수 있다는 장점을 얻을 수 있다. 그리고 다양한 GALT 계산 알고리즘을 각각의 GALT 모듈로 개발하여 교체함으로써 다양한 알고리즘들의 성능을 비교해볼 수 있다.

4.3 TIME 모듈

앞서 설명한 대로 TIME 모듈은 페더레이트의 정책을 결정하고 시간 진행 요청 서비스를 처리해준다. TIME 모듈을 설계하기 위해 모든 시간 관리 서비스 API에 대해 Unified Modeling Language(UML) 2.0의 시퀀스 다이어그램으로 모델링한 후, Discrete Event Systems Specification(DEVS)의 상태천이도로 표현하여 구현하였다. 그림 6은 LRC의 TIME 모듈의 상태천이도이다. 시간 진행 요청 서비스 중 하나인 Time Advance Request(TAR)를 예를 들어보면 다음과 같다.

(1) 사용자는 RTI ambassador에 *TAR*을 호출한다.
(2) 예외사항을 체크한 후, RTI ambassador는 Time AdvancePending 플래그를 설정한다.
(3) RTI ambassador는 LRC의 TIME 모듈과 TSO 큐로 TIME_ADVANCE 메시지를 보낸다.
(4) TIME 모듈은 IDLE 상태에서 TAR_WAIT 상태로 천이한다.
(5) TSO 큐는 GALT 보다 작은 Time-Stamp를 가지는 메시지를 전달한다.

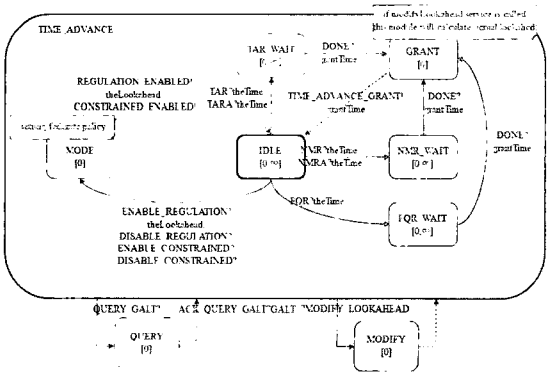


그림 6. LRC의 TIME 모듈의 상태천이도

(6) 시간 진행 요청된 값이 GALT보다 작으면 TSO 큐는 TIME 모듈로 DONE 메시지를 보낸다. 그렇지 않다면, GALT가 요청된 시간보다 커질 때까지 기다린다.
(7) TIME 모듈이 DONE 메시지를 받으면 페더레이트 ambassador로 TIME_ADVANCE_GRANT 콜백 메시지를 보낸다.
(8) 페더레이트 ambassador는 사용자에게 *Time Advance Grant*(TAG) 콜백을 보낸다.

4.4 GALT 모듈

GALT 모듈은 TIME 모듈과는 독립적으로 GALT 계산을 수행한다. 수정된 Mattern의 알고리즘을 구현하기 위해 중앙식 구조로 설계하였다. FedExec가 중앙 컨트롤러로서 컨트롤 벡터 카운터 C를 가지며 GALT 계산의 시작을 모든 페더레이트에게 알려주는 역할을 담당한다. 각각의 페더레이트 F_i는 두 개의 벡터 카운터 V_{i1}과 V_{i2}를 가지며 V_{i1}은 이전 컷에서 보내고 받는 메시지를 세고 V_{i2}는 이전 컷과 현재 컷 사이에서 보내고 받는 메시지를 센다. GALT 계산 과정을 살펴보면 아래와 같으며 그림 7은 이런 절차를 UML 2.0의 시퀀스 다이어그램으로 표현한 것이다.

(1) FedExec가 주기적으로 GALT 계산을 시작하기 위해 모든 페더레이트로 REQ_CTRL_MSG 메시지를 보낸다.
(2) 각 페더레이트는 컷 지점을 설정하고 컷을 통과하는 메시지를 센다.
(3) 각 페더레이트는 REQ_CTRL_MSG 메시지에 실려 있는 GALT 플래그가 1이면 자신을 제외한 페더레이트들의 시간 값 중 최소값을 새로운 GALT 값으로 계산한다.
(4) 페더레이트 F_i의 벡터 카운터 $V_i[i]$가 컨트롤 벡터 카운터 C[i]보다 작거나 같으면, 페더레이트 F_i는

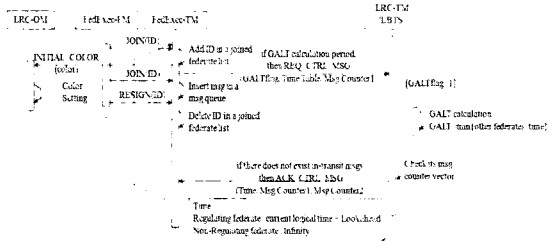


그림 7. GALT 계산의 시퀀스 다이어그램

FedExec로 ACK_CTRL_MSG 메시지를 보낸다. 그렇지 않으면 $V_i[i]$가 C[i]보다 작거나 같아질 때까지 기다린다. 이는 Mattern의 알고리즘에서 다음 프로세서로 컨트롤 메시지를 넘기는 것과 같은 조건이다.

(5) 페더레이트 F_i는 V_{i1}과 V_{i2}를 교환하고, V_{i2}를 0으로 초기화한다.

(6) FedExec가 모든 페더레이트로부터 ACK_CTRL_MSG 메시지를 받으면 메시지의 정보를 통합한다.

(7) C와 모든 V_{i1}의 합이 0이면 GALT 플래그를 1로 설정한다.

(8) FedExec는 REQ_CTRL_MSG 메시지의 정보를 갱신하고 모든 페더레이트에게 설정된 주기가 되면 GALT 계산 시작을 알리는 REQ_CTRL_MSG 메시지를 전송한다.

위의 과정 중 (4)에서 페더레이트가 FedExec로 보내는 ACK_CTRL_MSG 메시지는 페더레이트의 현재 시간, 벡터 카운터 V_1, V_2를 담고 있다. 이때 레귤레이팅 페더레이트이면 현재 시간 + Lookahead 값을 보내고, 그렇지 않으면 무한대의 값을 보낸다.

Mattern과 달리 중앙식 구조를 취함으로써 페더레이트에 도착하는 컨트롤 메시지가 순차적으로 도달하는 것이 아니라 모든 페더레이트에 동시에 도달한다. 중앙식 구조로 모델링함으로써 얻을 수 있는 이점은 설계의 복잡도가 줄어든다는 것과 페더레이트의 정보 관리가 용이하다는 것이다. 또한, 시뮬레이션 도중에 페더레이트가 참가하거나 탈퇴하는 경우에 대해서도 처리가 간편해진다. GALT 계산이 시작된 이후에 페더레이션에 참가하고자 하는 페더레이트는 바로 참가하지 못하고 현재 진행 중인 GALT 계산이 끝난 이후에 참가하도록 한다. 이는 늦게 참가하는 페더레이트가 현재 시뮬레이션되고 있는 페더레이트들과 같은 색깔을 띄어야하기 때문이다. 시뮬레이션 도중에 탈퇴하고자하는 페더레이트는 언제든지 탈퇴할 수 있다. 이는 페더레이트가 탈퇴하고자 할 때 FedExec에게 알림으로써 이 페더레이트와 관련된 정보를 모두 삭제하도록 하면 되기 때문이다.

5. 실험 결과 및 분석

본 논문에서 제안한 시간 관리 서비스를 모듈화하여 설계하고 구현한 시간 관리 서비스 모듈의 성능을 알아보기 위하여 수행한 실험에 대한 결과를 살펴본다. 7대의 컴퓨터로 실험 환경을 구성하였으며 1대에는 RTI와 페더레이션을 수행시키고 나머지 6대에 각각의 페더레이트 프로그램을 수행시켰다. 그림 8은 실험환경을 보여주고 있다. 실험에 참여하는 페더레이트는 모두 레귤레이팅이면서 컨스트레인드 정책을 취하고 있다. 실험은 GALT 계산 주기, Lookahead, 참여하는 페더레이트의 수, 페더레이트가 주고받는 메시지의 수를 실험의 파라미터로 사용하여 수행하였다.

페더레이션에서 각각의 페더레이트는 시간 진행을 위하여 *TAR* 서비스를 이용하고 TSO 메시지 전송을 위해 *Send Interaction* 서비스를 사용하며 메시지는 Reliable 방식으로 전송한다. 시간 진행 요청에서 시간 간격은 2이며 Lookahead는 1.9를 기본값으로 설정하였다. 실험에서 측정한 값은 *TAR* 서비스를 호출한 뒤 *TAG* 콜백을 받기까지의 시간이다.

시간 관리 서비스 모듈의 성능은 GALT 계산 주기에 큰 영향을 받는다. 그림 9(a)에서 보듯이 GALT 계산을 자주 할수록 페더레이트가 시간 진행을 하지 못하고 기다리는 시간이 줄어든다. 또한, Lookahead가 증가할수록 다른 컨스트레인드 페더레이트가 시간 진행에 있어 허용되는 시간이 늘어날 것이므로 *TAR* 요청에 대한 *TAG*를 받는 시간이 단축될 것이다. 그러나 시뮬레이션 속도를 향상시키기 위하여 Lookahead 값을 한없이 증가시킬 수 있는 것은 아니다. Lookahead 값은 레귤레이팅 페더레이트가 보내는 TSO 메시지의 Time-Stamp에 영향을 미치게 되므로 시뮬레이션 상황에 따라 정해지는 값이기 때문이다. 그림 9(b)는 참여하는 페더레이트의 수와 주고받는 메시지 수가 늘어날수록 시뮬레이션 시간이 증가함을 보여준다. 페더레이트의 수가 증가할수록 GALT 계산을 위해 사용되는 컨트롤 메시지의 수가 증가하며 하나의 라운드로 과도 메시지를 해결할 수 있는 가능성이 줄어들 것이다. 그리고 보내는 메시지의 수가 증가하면 GALT 계산에 있어 과도 메시지의 수가 증가할 수 있다는 것을 의미한다.

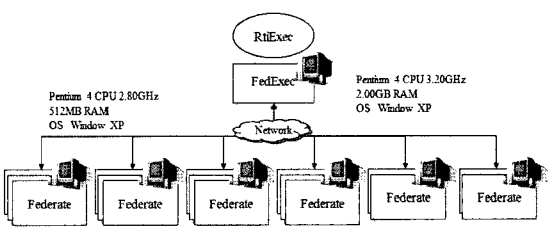
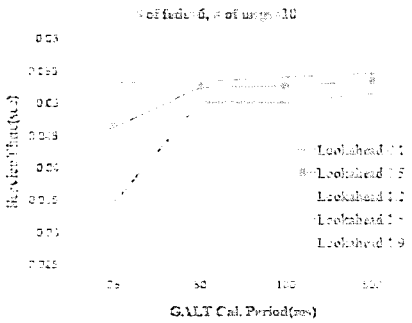
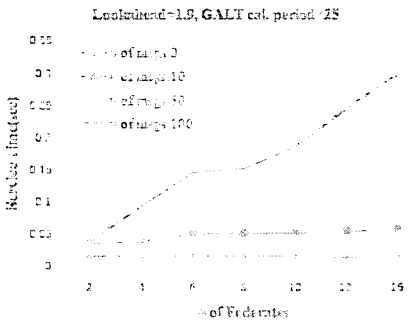


그림 8. 실험 환경



(a)



(b)

그림 9. 실험 결과

6. 결 론

본 논문에서 IEEE 1516 HLA/RTI의 6가지 서비스 중에서 시간 관리 서비스를 모듈화하여 설계하고 구현하는 방안을 제안하였다. 이로 인해 HLA/RTI의 시간 관리 API를 서비스하는 모듈을 재사용할 수 있으며 GALT 계산 모듈은 다른 알고리즘을 적용하기가 용이해진다. GALT 계산 알고리즘으로 분산 시뮬레이션에서 널리 사용되고 있는 Mattern의 알고리즘을 HLA/RTI에 적합하도록 수정·보완하였다. 또한 시간 관리 서비스 모듈의 설계를 위하여 내부 모듈간의 인터페이스뿐만 아니라 SMSRTI 내의 다른 모듈과의 상호작용을 위한 인터페이스를 정의하고 LRC와 FedExec 사이의 프로토콜도 정하였다. 개발된 시간 관리 서비스 모듈의 성능을 살펴보기 위하여 실험을 수행하고 그 결과 예상된 경향을 보임을 확인했다. 다양한 GALT 계산 알고리즘을 비교하여 최적화된 알고리즘을 적용시켜 시간 관리 서비스 모듈의 성능을 개선하는 것이 추후 과제로 남아있다.

감사의 글

본 논문은 과학재단 특정기초연구(R01-2006-000-111180)의 지원으로 "Collaborative BPMS 환경에서 모델링, 설계, 분석, 구축을 위한 BPR 도구로 활용될 수 있는 HLA/RTI 기반의 BPM middleware 및 요소기술 개발" 과제의 연구 결과이다.

참 고 문 헌

1. IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules, Std 1516, 2000.
2. IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification, Std 1516.1, 2000.
3. IEEE, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT), Std 1516.3, 2000.
4. J. H. Kim, "Proposal of High Level Architecture Extension and Run-Time Infrastructure Implementation," Ph. D. Thesis, KAIST, Daejeon, Korea, 2006.
5. F. Mattern, "Efficient Distributed Snapshots and Global Virtual Time Algorithm for Non-FIFO System," *Journal of Parallel and Distributed Computing (JPDC)*, Vol. 18, No. 4, pp. 423-434, August, 1993.
6. L. Lamport, "Time, Clocks and Ordering of Events in a Distributed System," *Comm. Of the ACM 21*, pp. 558-565, July, 1978.
7. Defense Modeling and Simulation Office (DMSO), High Level Architecture Run-Time Infrastructure RTI 1.3NG Programmer's Guide, Version 5, 1999.
8. R. M. Fujimoto, *Parallel and Distributed Simulation Systems*, John Wiley and Sons, Inc. 2000.
9. B. Samadi, "Distributed Simulation, Algorithms and Performance Analysis," Ph. D. Thesis, University of California, Los Angeles (UCLA), 1985.
10. Y-B. Lin and E. D. Lazowska, "Determining the Global Virtual Time in a Distributed Simulation," *in Proc. of the 1990 International Conference on Parallel Processing*, Vol. 3, pp. 201-209, August, 1990.
11. Pitch Technologies, 2006, http://www.pitch.se/prti.htm.
12. MÄK Technologies, 2006, http://www.mak.com/rti.htm.
13. R. M. Fujimoto, "FDK User Guide," Version 3.0, 2003, http://www.ee.gatech.edu.

홍 정 희 (jhhong@smslab.kaist.ac.kr)

2005 부산대학교 전자전기정보컴퓨터공학부 학사
2007 KAIST 전자전산학과 석사
2007～현재 KAIST 전자전산학과 박사과정

관심분야 : HLA/RTI, Time Management, Distributed Simulation

안 정 현 (jhahn@smslab.kaist.ac.kr)

2005 부산대학교 전자전기정보컴퓨터공학부 학사
2007 KAIST 전자전산학과 석사
2007～현재 KAIST 전자전산학과 박사과정

관심분야 : HLA/RTI, Data Distribution Management, Distributed Simulation

김 탁 곤 (tkim@ee.kaist.ac.kr)

1975 부산대학교 전기공학과 학사
1980 경북대학교 전기공학과 석사
1988 Univ. of Arizona, 전기및컴퓨터공학과 박사
1980～1983 부경대학교 통신공학과 전임강사
1987～1989 (미)아리조나 환경연구소 연구엔지니어
1989～1991 Univ. of Kansas 전기및컴퓨터공학과 조교수
1991～현재 KAIST 전자전산학과 교수

관심분야 : 모델링/시뮬레이션 이론, 방법론 및 환경개발, 시뮬레이터 연동