Journal of The Korea Society of Computer and Information
Vol. 17, No. 12, December 2012
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci/2012.17.12.011

BlockSim++: 연속시스템의 계층적 모델링 및 시뮬레이션을 위한 블록기반 경량 프레임워크

송 해 상*, 서 정 만**

BlockSim++: A Lightweight Block-oriented Hierarchical Modeling and Simulation Framework for Continuous Systems

Hae-Sang Song*, Jeong-Man Seo**

요 약

본 논문은 실용 엔지니어를 위해서 일반미분방정식으로 표현될 수 있는 연속시스템의 계층적인 모델 개발을 위한 모델링 기법 및 객체지향언어 기반의 경량 시뮬레이션 구축 환경을 제안하였다. 제안된 블록 기반 모델링 형식론은 단위 모델의 동적인 행위를 나타내는 기본블록 모델과 모델들의 계층적인 구조를 나타내는 결합블록 모델 형식론으로 이루어져 있다. 이러한 수학적 모델의 시뮬레이터를 객체지향 언어로 구현하기 위한 시뮬레이션구축 프레임워크인 BlockSim++를 제안하였다. 제안된 프레임워크는 재사용성을 제공하며, 수학적 모델을 쉽게 구현할 수 있게 하고, 또한 외부 응용 소프트웨어와도 쉽게 결합할 수 있는 인터페이스를 제공한다. 간단한 하이브리드 모델링 시뮬레이션 예시를 통해 제안된 모델링 형식론과 시뮬레이션 프레임워크를 이용하여 그 유용성을 입증하였다.

▸ Keywords : 연속 시스템, 블록 기반 연속시스템 모델링 시뮬레이션 환경, 내장형 시뮬레이터

Abstract

This paper proposes for practical engineers a lightweight modeling and simulation environment for continuous system models specified in ordinary differential equations, which are time-domain specification of such systems. We propose a block-oriented specification formalism that has two levels: one for atomic behavior and the other the structure of models. Also we provide with a simulation framework, called BlockSim++, which make models specified in the block-oriented

• 제1저자 : 송 해 상 • 교신저자 : 서정만
• 투고일 : 2012. 10. 11, 심사일 : 2012. 10. 18, 게재확정일 : 2012. 11. 6.
* 서원대학교 컴퓨터공학과 교수(Dept. of Computer Engineering, Seowon University)
** 한국복지대학교 게임콘텐츠과 교수(Dept. of Game Contents, Korea National College of Welfare)

formalism be easily translated in object-oriented program that runs with the proposed simulation framework. The proposed formalism and framework has advantage of reuse such that it can be easily integrated into application programs and heterogeneous simulators. We illustrates the usefulness of the proposed framework by a simple hybrid modeling simulation example.

▸ Keywords : Continuous Systems, Block-oriented Modeling and Simulation, Embedded Simulator

I. 서 론

미분방정식으로 표현되는 연속시스템에 대한 시뮬레이션은 그동안 상용 미분방정식 해석도구인 수치해석 라이브러리나, 상용도구 등을 통하여 진행되어 왔고 해석 알고리즘도 잘 알려져 왔다. 하지만 현대에 있어 시스템의 복잡도가 커짐에 따라 연속시스템 뿐만 아니라 이산사건시스템, 디지털 시스템, 이산시간시스템 등 성격이 다른 시스템과의 하이브리드 시뮬레이션과 다양한 응용프로그램의 일부분에 포함되어 동작하는 경우가 상존한다[1][2]. 연속시스템을 해석하기 위한 상용 도구의 경우 비용측면 뿐만 아니라 인터프리터 형식으로 구현되어 있어 이러한 응용 소프트웨어에 내장되기 어려운 측면이 있으며 또한 다양한 기능이 부가됨에 따라 핵심 시뮬레이션 엔진 외에도 매우 방대하고 무겁게 되었다[3]. 또한 저작권 문제가 있어 이러한 응용 소프트웨어에 내장되기에는 부적절한 경우가 존재한다[4]. 또한, 공개되어 있는 미분방정식 수치해석의 경우에도 하나의 방정식이 아니라 여러 컴포넌트가 계층적이고 모듈러하게 존재하고 이들이 서로 결합되어 동작하는 복잡한 시스템을 모델링하고 시뮬레이션하기에 적합하지 않다[5]. 따라서 본 논문에서는 이렇게 복잡한 연속시스템의 모델링과 시뮬레이션을 C++프로그램 언어를 이용하여 비전문가도 쉽게 구현할 수 있는 새로운 모델링 형식론 및 프레임워크를 제시하고자 한다.

제시하고자 하는 프레임워크의 응용분야의 중요한 예는 국방 워게임 모델링 시뮬레이션이다. 일반적으로 워게임 모델은 모델의 충실도(fidelity)에 따라 전구급, 임무급, 교전급, 그리고 공학급의 네 단계로 구분한다. 일반적으로 교전급 이상의 모델은 이산사건 시스템으로 모델링되고, 공학급 모델은 교전급에서의 이산사건 모델의 각 사건에 대해 무기 체계 등 참여 객체의 물리적 움직임까지 상세하게 모델링하기 때문에 미적분 또는 이산시간 시스템으로 표현된다. 이러한 모델은 MATLAB/ Simulink[3]과 같은 상용도구를 사용하거나 HLA/RTI[6][9]와 같은 연동 미들웨어를 이용하여 구현해 왔다. 전자의 경우에는 손쉽게 모델링 및 검증을 해 볼 수 있는 반면 비용과 성능, 다른 소프트웨어에 쉽게 내장되어 통합하기 어려운 면이 있으며 후자의 경우는 RTI의 비용과 개발하는데 드는 노력이 큰 측면이 존재한다. 따라서 본 논문은 이러한 문제를 해소하고자 다음과 같은 요구사항을 만족하는 연속시스템 모델링 시뮬레이션 개발 플랫폼을 제안하고자 한다.

- 편의성 - 비전문가도 쉽게 코딩이 가능할 것
- 재사용성 - 계층적이고 모듈화 특징을 가질 것
- 객체지향성 - 객체지향 언어를 사용할 것
- 통합성 - 응용 프로그램에 내장하기 쉬울 것
- 확장성 - 기능의 확장성을 고려할 것

본 논문의 2장에서는 이론적 배경 및 연구목적을 다루고 3장에서는 블록 기반 연속시스템 모델링 형식론을 제안한다. 4장에서는 블록 모델의 시뮬레이션 기법에 대해 제안하며 5장에서는 객체지향 언어로 모델을 쉽게 구현할 수 있게 제공하는 블록기반 경량 구현 프레임워크를 제안한다. 6장에서는 간단한 예제를 통해 제안된 형식론 및 구현 기법을 적용하여 그 유용성과 타당성을 입증하고 결론을 맺고자 한다.

II. 이론적 배경

2.1 시스템의 분류

일반적으로 시스템은 시간에 따라 상태가 변화되며, 상태집합의 형태를 이산집합과 연속집합으로 구분하고, 상태변화가 일어나는 시간을 연속 및 이산시간으로 구분할 때 [표 1]과 같이 네 가지 종류의 시스템 형태로 분류할 수 있다[1,7]. 그 중 본 논문에서 다루고자 하는 것은 시간에 대해 연속적으로 상태가 변하는 연속시스템으로 일반적으로 미분방정식으로 모델링할 수 있는 연속시스템이다.

표 1. 동적 시스템의 분류 및 시뮬레이션 환경
Table 1. Taxonomy of Dynamic Systems

구분	연속시간	이산시간
연속 상태	**연속시스템**	**이산신호시스템**
	미분방정식 Laplace Transform	차분방정식 Z-transform
	전기전자회로	DSP(digital signal processing)
	MATLAB	MATLAB
이산 상태	**이산사건시스템**	**디지털시스템**
	DEVS, Timed Petri-net, Timed Automata	불 대수, 유한 상태기계 (FSA)
	공정제어, 워게임, 네트워크	논리회로, 컴퓨터
	DEVSim++	System-C / SPICE

일반적으로 선형 미분방정식의 해를 구하는 많은 해석적인 방법이 있지만, 비선형적인 방정식에 대해서는 시뮬레이션을 통하지 않고는 구하기 어렵다. 이러한 경우 연속시스템은 일반미분방정식(ODE: Ordinary Differential Equation)으로 표현되고, 다양한 시뮬레이션 알고리즘을 통해 시간에 대한 해를 구할 수 있다고 알려져 왔다[5].

2.2 일반미분방정식

하나의 독립변수를 가지고 그 변수에 대해 하나 이상의 미분방정식을 가지는 경우를 일반미분방정식이라고 한다. 일반적으로 시뮬레이션에서의 독립변수는 시간 t이다[5,7]. 모든 일반미분방정식은 식(1)처럼 일차 미분방정식의 벡터로 표현될 수 있다.

$$\frac{d\vec{q}(t)}{dt} = \vec{f}(\vec{q}(t), t) \tag{1}$$

여기서 $\vec{q}, \vec{f}$ 는 벡터이다. 즉, $\vec{q} = [q_1, q_2, \ldots, q_n]$이라 할 때 이를 풀어쓰면 식(2)와 같다.

$$\begin{aligned} \frac{dq_1(t)}{dt} &= f_1(q_1(t), q_2(t), \ldots, q_n(t), t), \\ \frac{dq_2(t)}{dt} &= f_2(q_1(t), q_2(t), \ldots, q_n(t), t) \\ &\cdots \\ \frac{dq_n(t)}{dt} &= f_n(q_1(t), q_2(t), \ldots, q_n(t), t) \end{aligned} \tag{2}$$

이러한 일반미분방정식은 항상 초기 값 문제가 존재하며 변수의 초기 값이 주어져야 한다. 일반미분방정식이 만약 선형적(linear)이라면 이는 해석적으로 풀어낼 수 있다. 만약 그렇지 않다면 모델을 선형식으로 근사화하여 해석적으로 풀어낸다. 그러나 많은 시스템은 선형성을 갖지 않기 때문에 컴퓨터를 통하여 수치해석(numerical analysis)으로만 해석할 수 있다. 그러나 수치해석 방법은 하나의 모델에서는 잘 적용되나 이러한 모델이 결합되어 동작하는 복잡한 시스템의 경우에는 잘 표현하기 어려우며, 또한 다른 응용 프로그램 또는 이기종의 시뮬레이터와 결합 및 내장되어 동작하기에는 충분하지 않다. 따라서 체계적인 접근을 위해서 다음 절에서는 블록 기반 연속시스템을 계층적으로 표현할 수 있는 형식론을 먼저 제안한다.

III. 블록 기반 연속 시스템 형식론

3.1 연속원자모델 (AM) 형식론

연속원자 모델(AM: continuous Aomic Model)은 더 이상 분해할 수 없는 연속 시스템의 컴포넌트로서 컴포넌트의 동적 모델을 나타낸다[2]. 연속원자모델은 입력변수와, 상태변수, 그리고 출력변수의 세 집합과 일반미분방정식을 적용한 상태천이함수, 그리고 출력함수로 이루어진다. 즉, 연속원자모델은 세 개의 집합과 두 개의 함수로 이루어진 식(3)과 같이 기술된다.

$$AM = < X_v, S_v, Y_v, f, g > \tag{3}$$

- X_v : 입력 변수의 집합,
- S_v : 상태 변수의 집합,
- Y_v : 출력 변수의 집합,

$f : S_v \times X_v \rightarrow S_v$: 연속상태천이 함수,

즉, $\frac{d}{dt} S_v = f(S_v(t), X_v(t), t)$

$g : S_v \times X_v \rightarrow Y_v$: 연속출력 함수,

즉, $Y_v(t) = g(S_v(t), X_v(t), t)$

입력, 출력, 상태 변수는 여러 개의 변수들의 집합이므로, 연속상태천이 함수와 연속출력 함수는 식(2)와 같은 벡터 형식으로 기술된다. 여기서 각 함수는 반드시 선형일 필요는 없다.

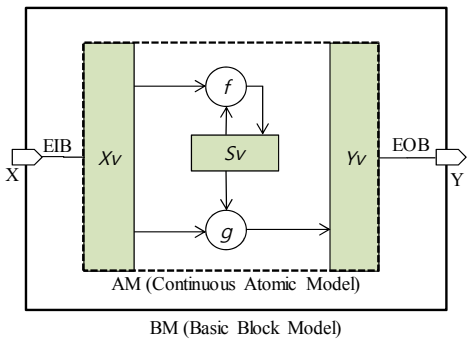

그림 1. 기본블록모델(BM) 형식론 개념도
Fig. 1. Concept of basic block model formalism

3.2 기본블록 모델(BM) 형식론

본 논문에서 제안하는 기본블록모델(Basic Block Model)은 다른 모델과의 결합을 다루기 위해서 연속원자모델을 감싸는 모델로서 식(4)와 같은 형식으로 기술된다.

$$BM = < X, Y, AM, EIB, EOB > \quad (4)$$

- X : 입력 포트의 집합,
- Y : 출력 포트의 집합,
- AM: 연속원자모델-식(3),
- EIB: $X \rightarrow X_v$: 입력포트 변수 접합 (binding) 함수,
- EOB: $Y \rightarrow Y_v$: 출력 포트 변수 접합 (binding) 함수

연속원자모델을 감싸고 있는 기본블록은 입력포트 X와 연속원자모델의 입력변수를 연결하고, 출력포트와 연속원자모델의 출력 포트를 접합(bind)하는 역할을 한다. 즉, 기본 블록은 동적인 연속 모델에 입출력 포트를 접합시켜 외부로 신호를 노출시켜 다른 모델들과 결합을 시키기 위한 것이다. 일반적으로 포트는 이름과 데이터타입을 갖는다. 이름은 연결하기 위한 것이고, 데이터타입은 해당포트를 통해 흐를 수 있는 연속신호의 구조를 기술하기 위한 것이다. [그림 1]은 기본블록 모델의 개념을 설명하고 있으며, 중요한 것은 입력포트와 연속원자모델의 입력변수, 출력포트와 연속원자모델의 출력변수를 연결하고 있는 모습을 나타내고 있다. 가운데의 연속원자모델은 입력, 출력, 상태변수로 구성되어 있고, 상태천이 함수 f는 입력변수와 상태변수의 값에 의해 다음 상태 변화를 나타내는 도함수이고, 출력함수 g는 현재 상태변수의 값과 현재 입력변수의 값에 의해 출력변수 값이 결정되는 모습을 표현한다.

3.3 결합블록모델(CM) 형식론

본 논문에서 제안하는 결합블록 모델은 외부입력, 외부출력 포트를 가지고 내부에 기본블록 또는 결합블록을 컴포넌트로 가지고 있으면서 결합된 포트끼리 신호를 전달하는 역할을 담당한다. 결합블록은 다음의 식(5)와 같은 형식으로 기술한다.

$$CM = < X, Y, M, EIC, EOC, IC > \quad (5)$$

- X : 외부 입력 포트들의 집합,
- Y : 외부 출력 포트들의 집합,
- M : 기본블록 또는 결합블록들의 집합,
- EIC: $\cup_i M_i.X_i \rightarrow X$: 외부 입력 결합함수,
- EOC: $Y \rightarrow \cup_i M_i.Y_i$: 외부 출력 결합합수,
- IC: $\cup_i M_i.X_i \rightarrow \cup_i M_i.Y_i$: 내부 결합함수

여기서 결합은 관계가 아닌 함수로 표현되었음을 유의하여야 하며, 이는 어느 한 포트로 입력되는 연속신호의 원천(소스)은 반드시 하나로 제약하는 것이다. [그림 2]는 결합모델의 개념도로서 두 개의 기본블록 BM-A와 BM-B가 결합되어 있는 모습을 나타낸 것이다. 기본블록 및 결합블록의 연결관계를 고려하고, 블록 모델을 제외하면, 연속원자모델의 출력변수가 다른 연속원자모델의 입력변수와 연결되어 있는 모습이라는 것을 쉽게 알 수 있다. 기본블록과 결합블록은 단지 신호의 연결만을 담당하기 때문에 실제로는 연속원자모델의 출력변수와 다른 연속원자모델의 입력변수를 연결하는 역할을 한다. 또한, 결합블록은 계층적으로 구성할 수 있어서 여러 블록을 연결시켜 하나의 블록으로 추상화할 수 있는 능력을 제공하여 재사용성의 편리함을 제공한다.

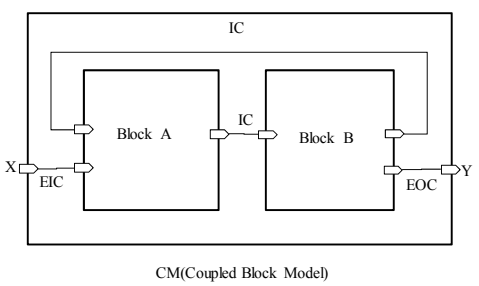

그림 2. 결합블록모델 형식론 개념도
Fig. 2. Concept of coupled block model formalism

Ⅳ. 블록기반 시뮬레이션 알고리즘 설계

4.1 연속원자모델(AM) 시뮬레이션 알고리즘

연속원자모델의 시뮬레이션은 Rung-Kutta 4 알고리즘이나 Euler, RK2 등 많은 알고리즘이 제시되었다[5]. 일반적으로 이러한 일반미분방정식의 수치해석적 해는 어느 한 시점의 상태로부터 다음 시점의 상태를 유추하는 적분을 위해 기울기를 사용하며 일반적으로 다음과 같이 세 개의 주요 기능으로 구현된다.

- Step 함수: 상태 x(t+dt)를 x(t) 및 기울기 dx/dt 도함수로부터 근사하여 스텝별 델타 값을 얻는 함수
- Integrate 함수: 얻어진 델타 값들의 가중치를 취해 적분 값을 얻어 x(t+dt)를 구하는 함수.
- UpdateClock 함수: step별 시간 증가함수

예를 들어 Runge Kutta 4 알고리즘은 시간 t에서 q값을 알고 있다고 가정할 때, 시간 t+h에서의 q값을 q′=f() 도함수, 즉 기울기를 이용하여 근사치로 적분하는 것으로 다음과 같이 기술된다. 먼저 도함수와 초기 값은 다음과 같이 주어진다고 가정한다.

$$\frac{dq}{dt} = q' = f(q,t), q(t_0) = q_0 \quad (6)$$

즉, q값의 변화는 q와 시간 t의 함수이며, 최초의 시간에 q값은 q0와 같다. RK4알고리즘은 다음과 같이 정의된다.

$$q_{n+1} = q_n + (k_1 + 2k_2 + 2k_3 + k_4)/6 \quad (7)$$

$$t_{n+1} = t_n + h \quad (8)$$

여기서 q_{n+1}은 $q(t_{n+1}) = q(t_n + h)$의 근사 값이고 h는 시간 간격으로 작을수록 정밀한 값을 얻을 수 있다. RK4에서 각 델타 값은 다음과 같이 정의된다.

$$k_1 = h \bullet f(q_n, t_n) \quad (9)$$

$$k_2 = h \bullet f(q_n + k_1/2, t_n + h/2) \quad (10)$$

$$k_3 = h \bullet f(q_n + k_2/2, t_n + h/2) \quad (11)$$

$$k_4 = h \bullet f(q_n + k_3, t_n + h) \quad (12)$$

여기서 식 (9)-(12)는 4개의 델타 값 k_i을 구하기 위한 Step 함수들이고 이렇게 구해진 델타 값들의 가중평균을 내어 다음 시간의 상태변수 q의 값을 구하는 식 (7)가 적분(Integrate)함수이다. 즉 다음 q의 값(q_{n+1})은 현재 값(q_n)과 델타(k_i) 값들의 가중평균의 합으로 결정된다. 위에서 각 델타 값은 적분시간간격 $h = \Delta t$와 추정된 기울기의 곱이다. RK4알고리즘을 활용한 적분에서는 4번의 마이크로 스텝을 거쳐서 네 개의 증분 값을 구한 후에 최종적으로 가중평균값을 구하고 있다. 여기서 우리는 단위 적분시간 간격 내에서 도함수 f에 대해 네 번의 계산이 일어나야 함을 알 수 있다. 만약 상태변수가 벡터로 주어질 때에는 도함수는 모든 상태변수들과 시간에 대한 함수로 주어지기 때문에 식 (6)의 함수 f가 식 (2)와 같이 모든 변수에 대한 함수로 확장 표현되고, 각 상태변수들에 대해서 도함수를 수행하는 것으로 확장된다. 각 마이크로 스텝별 시간 증가는 식(9)-(12)에 나타나 있다.

4.2 블록 모델 시뮬레이션 알고리즘

식(4)의 기본블록 형식론과 식(5)의 결합블록 형식론을 보면 기본블록이 감싸고 있는 식(3)의 연속원자모델의 출력변수를 포트간 연결를 통하여 다른 연속원자모델의 입력변수들과 연결됨을 알 수 있다. 이를 위한 블록 시뮬레이션 알고리즘은 이러한 연결된 입출력 변수들간의 동기화만을 담당한다. 즉, 블록 시뮬레이션 알고리즘은 모든 원자모델의 입력변수와 접합된 입력포트와 연결된 다른 원자모델의 출력 포트를 찾아, 접합된 출력변수의 값을 읽어서 그 입력변수에 복사하는 것이다. 따라서 각 마이크로 스텝마다, 모든 출력변수들의 값들이 연결된 모든 입력변수들에 한 번씩 복사되는 쉐도우(shadow) 변수의 개념을 가지게 된다.

4.3 전체적인 시뮬레이션 실행 알고리즘

4.1절과 4.2절을 만족하는 실행 알고리즘은 [그림 3]과 같다. 이 알고리즘은 기본블록과 결합블록 모델이 시뮬레이션 커널에 등록되어 있다고 가정한다. 또한 결합블록 모델은 평탄화되어 시뮬레이션 커널은 기본블록의 목록에 대해 적분 및 입출력 변수의 동기화를 일정시간 간격으로 시작에서 종료 시점까지 반복해서 수행한다.

Algorithm : BlockSim::Run(Time h, Time Start, Time End)

입력: h - 적분 시간 간격, Limit - 적분 구간

1. 모든 연속모델에 대해서 입력변수와 상태변수의 값 초기화 및 t = Start.
2. 모든 연속모델에 대해서 출력함수 수행 y = g(x,q,t)
3. 모든 연속모델의 입력변수에 대해서 각각 연관된 출력변수의 값을 복제, 즉, 모든 블록의 x에 대해서 x = y', 여기서 y'은 x과 포트로 연결된 출력변수
4. 각 연속모델이 가지는 모든 상태변수 q와 그 도변수 dq에 대해서 t부터 t+h 구간의 적분을 위한 마이크로스텝을 수행하여 각 스텝의 델타 값을 구함

4.a dq = fq(-), 즉 도함수를 수행하여 dq에 저장

4.b i 번째 델타 값 k_i를 구하는 DoStep() 함수 수행

5. 마이크로 스텝의 시간을 증가하는 UpdateClock() 함수 수행
6 적분 마이크로스텝이 남았으면 4부터 반복
7. 마이크로스텝이 끝났으면 델타 값을 가중 평균하여 다음 시각에서의 적분을 계산하여 q에 반영.
8. 시각을 t = t+h로 증가하고 만약 t < End이면 2부터 반복
9. 종료조건을 만족하면 시뮬레이션 결과 값을 반환

그림 3. 블록 모델의 시뮬레이션 알고리즘
Fig. 3. Simulation algorithm of block models

V. 시뮬레이션 프레임워크 설계

5.1 블록 모델 설계

블록 모델 시뮬레이션 모델은 크게 연속모델과 이를 감싸는 기본블록, 그리고 결합블록으로 구분된다. [그림 4]는 기본블록 모델에 대한 설계 개념도로서 연속원자모델 AM은 입력변수들과 출력변수들, 그리고 상태변수들을 선언하고 상태변환함수 f()와 출력함수 g()를 구현하게 된다. 이를 소프트웨어로 구현하기 위해 각 상태변수 q에 대한 도함수의 계산결과를 도변수 dq에 저장하는 것으로 설계한다. 즉 f() 함수가 수행될 때마다 도변수들에는 도함수의 수행결과가 저장된다고 가정한다.

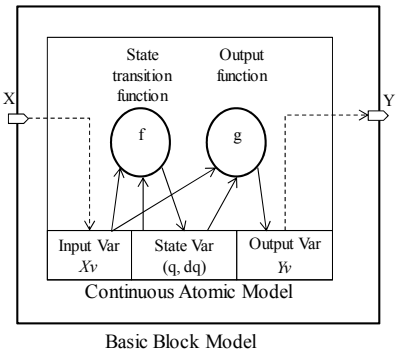

그림 4. 기본블록 모델 설계 개념도
Fig. 4. Design concept of a basic block model

이와 같은 개념을 토대로 기본블록 모델을 구현할 수 있도록 설계된 클래스도는 [그림 5]와 같다.

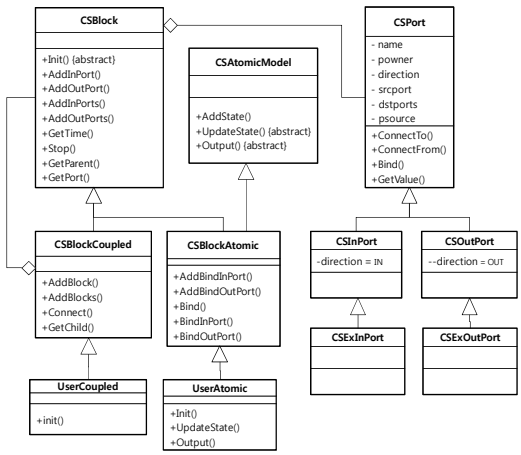

그림 5. 사용자 모델을 구현을 위한 기본 클래스도
Fig. 5. Class diagram for user block models

사용자는 좌측하단의 그림처럼 UserAtomic 및 UserCoupled 와 같이 BlockAtomic 및 BlockCoupled을 상속하여 사용자 모델을 구현한다. 제공되는 각 클래스의 접두어는 CS이다. 기본블록 형식론으로 모델링된 기본 블록 모델은 BlockAtomic 클래스를 상속하여 구현하는데 먼저 입력변수, 출력변수, 상태변수 및 상태도변수를 선언한다. Init()함수는 커널에 의해 시뮬레이션 초기에 호출되는 함수로서 입력 및 상태 변수의 초기 값을 설정한다. 기본블로모델의 상태변환함수 f()는 각 상태변수들에 대한 도함수들을 dq = f(x,q,t) 형태로 UpdateState() 함수 내에 구현한다.

여기서 가장 중요한 것은 생성함수에서 상태변수 q 및 도변수 dq의 관계를 AddState() 함수를 사용하여 등록하는 것이다. 이는 시뮬레이션 커널에게 두 변수와의 관계를 알려주어 적분하기 위함이다. 출력함수 Output() 내에서는 각 출력변수에 대해 y=g(x,q,t) 모델을 프로그래밍 언어로 기술하면 연속원자모델에 대한 구현이 완료된다. 마지막으로 이 모델을 둘러싸고 있는 기본블록을 통해 입출력변수의 값을 외부로 노출시키기 위해 AddInPort(), AddOutPort()함수를 이용하여 먼저 입출력 포트를 추가하고, 이를 입출력변수에 각각 묶어주는 Bind() 함수를 수행한다. 이렇게 하면 기본블록의 구현이 완료된다. 사용자의 결합블록 모델은 BlockCoupled를 상속하여 구현하며 생성함수에서 외부 입출력 포트를 추가하고 기 구현되었거나 기본으로 생성된 블록을 AddBlock() 함수를 사용하며 등록한다. 이 후 EIC, EOC, IC를 구현하기 위해 Connect() 함수를 사용하여 소스블록과 소스포트, 목적블록과 목적 포트를 지정한다.

5.2 시뮬레이션 커널 설계

시뮬레이션 커널의 개략적인 설계도는 〔그림 6〕과 같다. 커널의 내부까지 사용자가 알 필요는 없지만, 가장 상위 클래스인 BlockSim을 활용하는 법을 알 필요가 있다. BlockSim 클래스는 시뮬레이션을 총괄하는 제어 클래스로서 이를 통해 사용자가 앞에서 구현한 모델을 등록해 주어야 한다. 커널의 핵심인 적분기에 해당하는 State객체는 BlockAtomic으로부터 상속한 기본블록모델 객체에서 AddState() 함수에 의해 등록된 선언한 상태변수 및 도변수를 통해서 적분하는 알고리즘을 내장하고 있다. State 클래스의 고정함수인 UpdateClock()는 적분알고리즘에 따라 매 마이크로 스탭마다 시간을 증가시키고 DoStep()함수가 실행되는데, DoStep() 함수는 그 때마다 BlockAtomic 모델의 UpdateState() 함수를 호출하여 도함수를 실행시켜 상태도 변수에 저장시키고 이를 통해 델타 값을 구한다. 적분 스텝이 끝나면 Integrate() 함수를 통해 상태변수의 적분값을 구한다. State 객체는 Create 패턴을 사용하여서 생성되며, 커널에 내장된 기본적분알고리즘 RK4가 아닌 다른 알고리즘으로 확장할 수 있도록 SetIntegrator() 함수를 제공하여 적분기를 등록하도록 설계되었다.

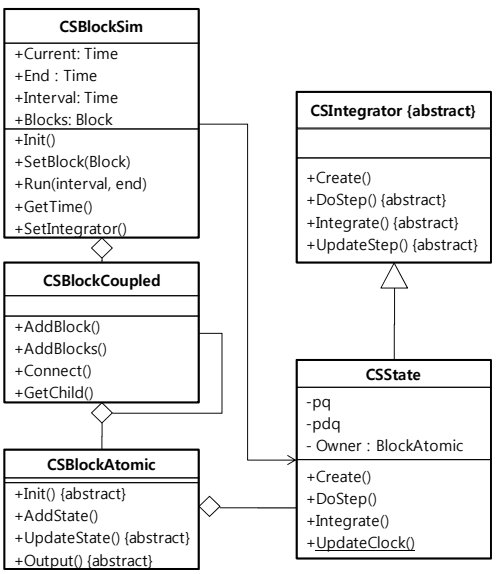

그림 6. 시뮬레이션 커널에 대한 클래스도
Fig. 6. Class diagram of simulation kernel

BlockSim의 Init() 함수는 시뮬레이션을 준비하는 함수로서 커널은 성능을 개선하기 위하여 계층화된 모델을 계층을 없애는 평탄화 (flattening) 작업을 통하여 BlockAtomic 의 목록과 포트간 연결 목록을 만든 후 모든 블록의 Init()을 수행하여 변수를 초기화한다. 〔그림 3〕에서 기술한 알고리즘을 구현한 Run() 함수는 시작 시각부터 정해진 시간간격마다 적분을 수행한다. 적분알고리즘은 모든 State 객체에 대해서 DoStep()을 반복하고, DoStep()함수에서는 BlockAtomic의 UpdateState() 함수 및 Output() 함수를 반복적으로 호출하면서 지정된 시간까지 시뮬레이션이 성공했거나 에러가 발생하면 반환한다. GetTime()함수는 호출할 때까지 진행된 시뮬레이션 시간을 반환한다.

5.3 외부 프로그램과의 데이터 인터페이스 설계

블록 모델간의 데이터 교환은 포트 개념을 사용한다. 특히 연속시스템 시뮬레이터와 외부 프로그램과 데이터를 교환하는 것은 외부 입출력 포트인 〔그림 5〕의 ExInPort 및 ExOutPort 클래스를 사용한다. 각 포트는 입출력 변수와 접합되어 있는 것은 시뮬레이션 모델의 것과 동일하지만 특별하게 외부 소프트웨어에 선언되어 있는 외부 포트 객체는 커널이 특별하게 관리하기 위해 등록한다는 것만 다르다.

예를 들어 〔그림 7〕에서 시뮬레이터와 결합하는 응용 소프트웨어인 클라이언트 소프트웨어의 변수 yc는 "cout"포트를 통해서 블록모델 M2의 "bin" 입력 포트와 연결되고, "bin"은 내부입력변수 x와 연결된다. 블록 시뮬레이터가 동작하게 되면 "bin"포트와 연결 관계를 찾아가서 궁극적으로 yc 변수의 값을 읽어 M2의 변수 x에 복사하는 동작을 매 사이클마다 반복하여 동기화한다. 반대로 M1 모델의 y 변수 값은 클라이언트가 선언한 "cin" 포트에 연결된 xc 변수로 동기화되게 된다.

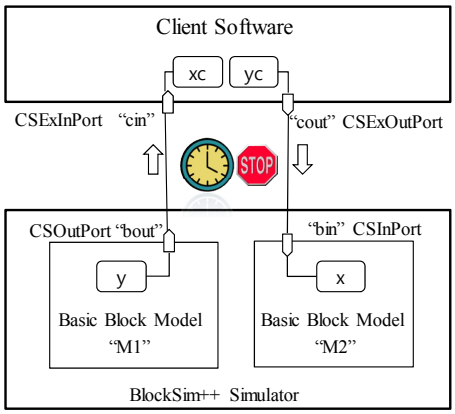

그림 7. 외부 소프트웨어와의 인터페이스
Fig. 7. Interface for external application software

Ⅵ. 실 험

6.1 물탱크 제어 시스템 예제

제안된 프레임워크를 간단한 물탱크 수위 조절 시스템의 시뮬레이션에 적용하여 보았다. [그림 8]은 제어기와 물탱크의 두 개의 부시스템으로 구성되어 있다. 물탱크에는 두 개의 수위 센서 (만수, 부족)가 있으며, 감지된 물탱크의 상태는 제어기에 전달된다. 제어기는 센서로부터 상태를 전달받아 ON/OFF 스위치를 통해 제어 신호를 펌프에 전달하여 물이 항상 만수와 부족 경계선 사이에 위치하도록 제어한다. 위 예제는 참고문헌 [8]의 HLA/RTI 연동 하이브리드 시뮬레이션의 예를 차용하였으며 이는 시뮬레이션 결과의 비교를 위한 것이다.

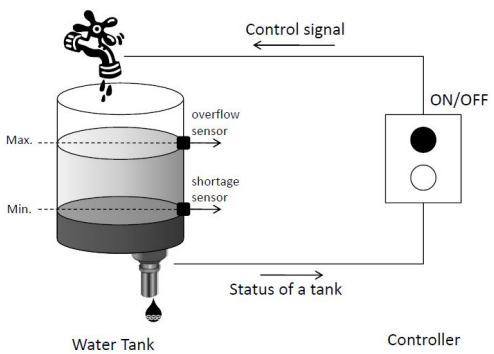

그림 8. 물탱크 수위 제어 시스템 예제
Fig. 8. Example: Water tank control system

물탱크 부시스템은 수위에 따른 수압에 비례한 배수와 펌프를 통한 입수의 연속시스템으로 모델링될 수 있다. 물이 탱크로 들어가는 유속은 펌프에 입력되는 전압 V에 비례하고 물이 배수구를 통해 나가는 유속은 탱크의 물의 높이 H의 제곱근에 비례한다고 할 때 일반 미분방정식은 식(13)과 같이 주어진다[5].

$$A\frac{dH}{dt} = bV - a\sqrt{H} \tag{13}$$

위에서,

V : 펌프의 입력 전압,

A : 탱크의 단면적,

b : 탱크의 입수 유속 계수,

a : 탱크의 배수 유속 계수

식(13)에서 입력변수는 입력전압 V, 상태변수는 수위 H, 출력변수도 수위 H이다.

6.2 기본블록 모델 설계 및 구현

위 식에 (13) 대한 일반적인 블록도는 [그림 9]에 도시되었다. 입력포트voltage 및 출력포트 height와 inflow가 연결되어 있다. 세 포트는 각각 입력변수 V, 출력변수 H 및 P와 접합되었다.

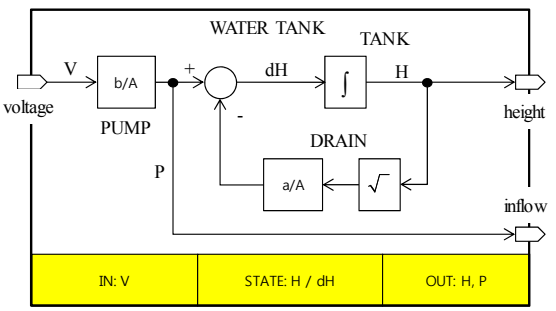

그림 9. 물탱크 연속 모델의 블록도
Fig. 9. Block diagram of water tank model

수학적으로 물탱크의 연속 모델 $AM-WT=<X_v,S_v,Y_v,f,g>$ 은 다음과 같이 정의된다. 입력변수 집합 $X_v=\{V\}$, 상태변수 집합 $S_v=\{H\}$, 출력변수집합 $Y_v=\{H\}$ 이며, 상태천이함수는 각 상태변수에 대한 도함수들에 대한 벡터로 표현된다. 상태변수가 수위 H 하나이므로 $f(\bullet)=[f_H]$ 이며, 식(13) 도함수는 $H'=dH=f_H(H(t),V(t),t)$

$= b/A \bullet V(t) - a/A\sqrt{H(t)}$ 이다. 출력함수는 출력변수 H와 P에 대한 출력함수들의 벡터 $g(\bullet) = [g_H \ g_P]$이다. 여기서 $H(t) = g_H(H(t),, V(t), t) = H(t)$, $P(t) = g_P(H(t), V(t), t) = b/A \bullet V(t)$임을 알 수 있다.

위 원자모델 AM-WT를 포함하고 있는 기본블록 $BM-WT = < X, Y, AM-WT, EIB, EOB >$는 다음과 같이 기술된다. 입력포트집합 X = {voltage}, 출력포트집합 Y = {height, inflow}이며, 입력결합 함수 $EIB: X \rightarrow X_c$를 관계 집합으로 표현하면 EIB = {(V, voltage)}와 같고, 마찬가지로 출력결합 함수는 $EOB: Y \rightarrow Y_c$는 EOB = {(H, height),(P,inflow)}로 나타낼 수 있다. 각 변수에 대한 초기 값은 V=0.0, H=0.0으로 가정하고, 상수 A=20.0, a=2.0, b=4.0으로 가정한다. 이러한 수학적 모델에 대한 구현 코드의 예는 다음과 같다.

```
class WaterTank : CSBlockAtomic
{
private:
        double H, dH, V; // I/O/S variables
        double A, a, b;    // coefficients
public:
        WaterTank()
        {
            AddBindInPort("voltage",V); // an input port
            AddBindOutPort("height",H);     // an output port
            AddBindOutPort("inflow",P);     // an output port
                AddState(H,dH);             // state, derivative
        }

        void Init(int nStage)
        {
            if (nStage == 0) // the initial stage
              A = 20.0, b = 4.0, a = 2.0, V = 0.0, H = 0.0;
        }
        void UpdateState()
        {
            dH = b/A*V - a/A*sqrt(H);    // eq. (10)
        }
        void Output()
        {
           H = H;         // output function for H
           P = b/A*V;    // output function for P

           cout << "Time:"   << GetTime();
           cout << H <<   " " << V << endl;
        }
}
```

위 코드는 모델과 비교해 볼 때 매우 직관적으로 이해될 수 있다. 생성함수는 객체의 생성 때 단 한번 호출되며, 입출력 포트를 만들고 입출력변수와 결합(bind)시킨다. 또한 상태변수와 도변수를 지정하여 적분이 가능하게 한다. 함수 Init()는 새로운 시뮬레이션 스테이지가 시작될 때 호출된다. UpdateState()함수는 적분알고리즘의 매 마이크로스텝마다 호출되는 것으로서 일반미분방정식을 그대로 구현한 것이다. 그러므로 매우 효율적으로 작성할 필요가 있다. Output() 함수는 매 적분 간격마다 호출이 되는 함수로서 출력변수를 갱신할 때 호출된다. 함수 Stop ()은 시뮬레이션을 종료(일시정지)하라는 의미이며 시뮬레이션 한 스텝이 끝날 때 매개변수 값 WT_CONTROL을 응용프로그램에 반환한다.

6.3 결합블록 모델 설계 및 구현

〔그림 10〕은 본 예시의 전체 구조도를 나타낸 것이다. 앞 절에서 기술한 물탱크 기본블록과, 수위 센서 기본블록, 두 모델을 결합한 물탱크 플랜트 결합블록과, 물탱크 플랜트 시뮬레이션 모델과 연동하는 응용 프로그램인 제어 소프트웨어로 이루어져 있다. 수위 센서 기본블록인 SENSOR 모델은 물탱크 기본블록의 출력포트 height와 센서 입력포트 level이 연결되었고, level 은 다시 WL 변수와 접합되어 있다. 센서의 출력은 HS 및 LS가 각각 high 포트 및 low 포트로 접합되어 값을 출력한다. 상태변수가 없으므로 변환함수는 없으며, 단지 출력함수만 존재한다. HS에 대한 출력함수는 입력된 수위 WL값이 OVERFLOW 보다 크면 ON을 그렇지 않으면 OFF를 출력한다. 마찬가지로 LS는 WL 값이

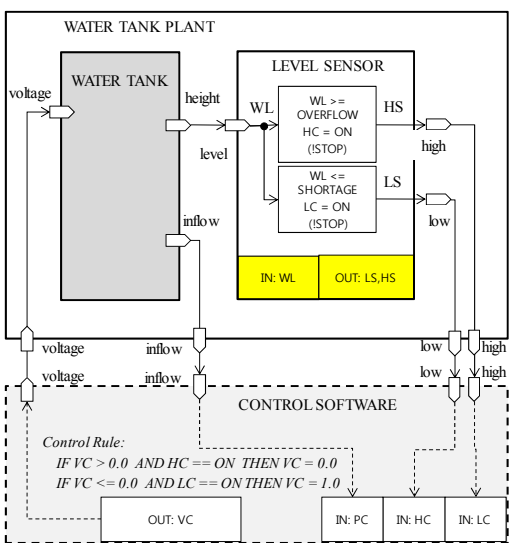

그림 10. 물탱크 플랜트 및 응용소프트웨어 인터페이스
Fig. 10. Application software interface with simulator

SHORTAGE보다 작거나 같으면 ON을 그렇지 않으면 OFF를 출력한다. 수위 센서에 대한 상태변수가 없으므로 UpdateState() 함수는 재정의되지 않아도 되며, 출력함수 Output() 만 정의된다. 먼저 수위 센서 SENSOR 기본블록의 모델은 다음과 같이 구현된다.

```
class Sensor : CSBlockAtomic
{
private:
        double WL, LS, HS; ; // I/O variables
public:
        LevelSensor ()
        {
            AddBindInPort("level",WL);// an input port
            AddBindOutPort("high",HS); // an output port
            AddBindOutPort("low",LS);// an output port
        }
        void Output()
        {
           if (WL > OVERFLOW)          HS = ON;
           else if ( WL < SHORTAGE)  LS = ON;
           else { HS = LS = OFF; return;}

           Stop(WT_CONTROL);
        }
}
```

물탱크 모델과 센서 모델을 결합한 물탱크 플랜트 결합블록의 수학적 모델 $CM-PLANT=<X,Y,M,EIC,EOC,IC>$는 다음과 같이 정의된다. X = {voltage}, Y={inflow, low, high}, M={TANK, SENSOR}, EIC ={(TANK.voltage, voltage)}, EOC={high, SENSOR.high), (low, SENSOR.low)}, IC = {(SENSOR.level, TANK.height)}. 이 결합블록의 구현 모델의 예는 다음과 같다.

```
class WaterPlant : CSBlockCoupled
{
private:
        CSBlock *theTank, *theSensor;
public:
        WaterPlant()
        {
            AddInPort("voltage");  // X
            AddOutPorts(3, "high",  "low", "inflow"); // Y
            theTank = new WaterTank();
            theSensor = new Sensor();
            AddBlocks(2, theTank, theSensor); // M
        }
        void Init(int nStage)
        {
            if (nStage == 0) { // the initial stage
                Attach(); // lazy coupling
            }
        }
        void Attatch()
        {
           Connect (this,"voltage",theTank, "voltage"); //EIC
           Connect (this, "low", theSensor,"low");     //EOC
           Connect (this, "high", theSensor, "high");  //EOC
           Connect (theTank,"height",theSensor,"level"); //IC
        }
}
```

6.4 외부 응용 소프트웨어 설계 구현

제어 소프트웨어는 높이를 입력으로 받아 HC변수에 저장하고, 이를 바탕으로 입력 전압을 결정하는 규칙을 가지고 비선형적으로 동작한다.

VC = 0.0 if H >= OVERFLOW until H > SHORTAGE
VC = 4.0 if H <= SHORTAGE until H < OVERFLOW

외부 소프트웨어에서 플랜트 모델의 출력포트 high 및 low와 연결하기 위하여 입력포트 high 및 low 포트를 만들고, 또한 플랜트로 제어 전압을 보내기 위해 voltage 포트를 만든다. 각 포트는 내부 변수들과 접합된다. 제어를 위해 일반적인 클래스 WaterTankController를 만들고 제어에 알고리즘을 Control() 함수에 구현한다. 제어 소프트웨어는 C++로 다음과 같이 구현할 수 있다.

```
class WaterTankController // an ordinary class
{
        double VC, HC, LC, PC;
        WaterPlant *theWaterPlant;

        CSExOutPort *pvoltage;
        CSExInPort  *phigh, *plow, *pflow;
public:
        WaterTankController(WaterPlant * aWaterPlant)
        {
            theWaterTank = aWaterPlant;

           *pvoltage = new CSExOutPort("voltage",VC);
           pvoltage->ConnectTo(
           theWaterPlant->GetInPort("voltage"));
           //...

           VC = 0.0, HC=LC=OFF;
        }
```

```
        void Control()
        {
            if (VC > 0   && HC == ON)    VC = 0.0;
            if (VC <= 0 && LC == ON)    VC = 1.0;
        }
}

void main()
{
        WaterTank *theWaterPlant = new WaterPlant();
        WaterTankController *theController = new
           WaterTankController(aWaterPlant);

        CSBlockSim *theSim = new CSBlockSim();
           theSim->AddBlock(theWaterPlant);

        theSim->SetSimTime(0.01, 250.0) ;
        theSim->Init();
        result = theSim->Run(); // simulation software
        while (result == WT_CONTROL) {   // Sim Event
            theController->Control(); // application software
            result = theSim->Run(); // continue simulation
        }
}
```

일반적인 제어 클래스 WaterTankController는 출력 변수 VC 및 입력변수 HC, LC, PC의 값은 시뮬레이션 커널에 의해서 연결되어 있는 모든 외부 포트의 값을 매 스텝마다 동기화시키므로, 상기 포트들에 접합된 변수들은 시뮬레이션 스텝이 끝날 때마다 연결된 시뮬레이션 모델의 변수들과 동일한 값을 가진다. 최초에 수행되는 main() 함수에서는 시뮬레이션 엔진인 CSBlockSimulator 객체를 하나 만들고 플랜트 모델을 추가하여 준다. 시뮬레이션 시간을 설정한 후 실행을 시킨다. 물탱크 기본 블록에서 Stop() 함수가 수행되면 시뮬레이터는 일시정지가 되어 Run()에서 반환하고 제어권을 외부 프로그램에 넘긴다. 반환 값이 WT_CONTROL이면 시뮬레이터에서 제어 응용 프로그램으로 제어가 필요하다는 예외가 발생했음을 의미한다. 응용 소프트웨어의 제어 함수를 한 번 수행해 준 후 다시 시뮬레이션을 계속하여 최대시간까지 완료되면 정상적으로 종료한다.

6.5 실험 결과

본 논문에서 제안한 BlockSim++ 프레임워크를 이용하여 상기 모델을 구현하고 시뮬레이션 하여 얻은 결과를 〔그림 11〕에 도시하였다. A = 20, a = 2, b = 4, H = 0로 초기 값을 부여하였고, 펌프를 구동할 때의 입력전압은 1로 정지할 때에는 0으로 주었다. 저수위 한계는 1로 고수위 한계는 4로 하였다. 적분시간간격은 0.01로 하였다. 이 결과는 동일한 조건하에서 상용도구인 SIMULINK 시뮬레이터 및 제어용 이산사건 시뮬레이터 DEVSim++와의 HLA연동을 통해 얻은 〔8〕의 결과와 정확하게 일치함을 알 수 있었다.

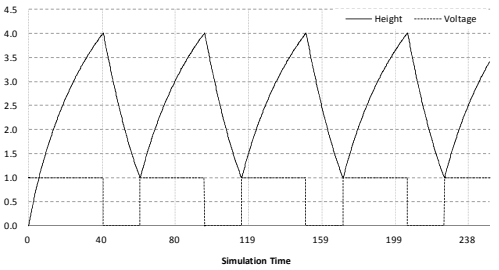

그림 11. 실험 결과: 수위 및 제어전압의 변화
Fig. 11. Result: variation of level and voltage

Ⅶ. 결론

본 논문은 연속시스템 모델을 계층적이고 모듈러하게 모델링할 수 있는 수학적 기반을 제안하였고, 이를 구현할 수 있는 시뮬레이션 프레임워크에 대한 프로토타입을 제안하였다. 또한 다른 응용 소프트웨어 또는 이산사건 시뮬레이터 등에 내장시켜 결합할 수 있게 하기 위한 인터페이스를 제안하였다. 제안된 모델링 방법론 및 구현 프레임워크의 타당성을 입증하기 위해 간단한 물탱크 모델링 및 시뮬레이션 구현, 그리고 그 결과를 MATLAB의 결과와 비교하여 올바른 결과를 얻었음을 보였다. 제안된 블록 기반 계층적 모델링 시뮬레이션 프레임워크는 모델의 재사용성 및 개발의 용이성을 제공하며, 응용 소프트웨어 및 다양한 하이브리드 M&S 환경에도 적용할 수 있을 것으로 기대된다.

참고문헌

〔1〕 B. P. Zeigler, T. G. Kim, and H. Praehofer, Theory of Modeling and Simulation, Orlando, FL: Academic, 2000.

〔2〕 S. Y. Lim and T. G. Kim, "Hybrid Systems Modeling and Simulation - Part I: Modeling and Simulation Methodology," Journal of Korea Simulation Society, vol. 10, no. 3, pp. 1-14, 2001.

〔3〕 M. Clune, P. Mosterman, and C. Cassandras, "Discrete Event and Hybrid System Simulation with SimEvents," in Discrete Event Systems,

2006 8th International Workshop on, Ann Arbor, MI, USA, Jul. 2006, pp. 386-387, 2006.

〔4〕 P. A. Hawley and R. A. Blauwkamp, "Six-Degree- of-Freedom Digital Simulations for Missile Guidance, Navigation, and Control," Johns Hopkins APL Technical Digest, vol. 29, no. 1, pp. 71-85, 2010.

〔5〕 John C. Butcher, Numerical methods for ordinary differential equations, John Wiley & Sons, 2003.

〔6〕 IEEE Std. 1516-2000. IEEE Standard for Modeling and Simulation(M&S) High Level Architecture (HLA) - Framework and Rules, IEEE Computer Society, 2000.

〔7〕 H.S. Song, "Survey of Modeling Simulation Methodologies and Classification of Dynamic Systems," Journal of Science & Culture, vol. 9, no. 1, pp. 101-118, 2012.

〔8〕 Chang Ho Sung and Tag Gon Kim, "Framework for Simulation of Hybrid Systems: Interoperation of Discrete Event and Continuous Simulators Using HLA/RTI," 25th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2011), Nice, France, June 14-17, 2011.

〔9〕 Jeong-Gook Koh, "Design and Implementation of a simulator for teaching disk scheduling algorithms", Journal of The Korea Society of Computer and Information. Vol. 16, No. 12pp. 131-137, Dec. 2011.

저 자 소 개

송 해 상

1991년 한국과학기술원 전기 및 전자공학과 공학석사
2000년 한국과학기술원 전자전산학과 전기및전자공학전공 공학박사
1999년~2000년 고등기술연구원
2002년~현재
서원대학교 컴퓨터공학과 교수
관심분야 : 시스템 모델링 시뮬레이션 이론 및 응용, 국방시스템 공학, 소프트웨어공학
Email: hssong@seowon.ac.kr

서 정 만

1988년 ~ 1993년 엘지전자 컴퓨터연구소 주임연구원
1993년 ~ 1999년 삼성중공업 중앙연구소 선임연구원
2000년 ~ 2002년 강동대학교 컴퓨터게임개발과 교수
2003년 충북대학교 컴퓨터공학과 박사
2002년 ~ 현재
한국복지대학교 게임콘텐츠과 교수
관심분야 : 실시간처리, 게임프로그래밍, 가상현실, 데이터베이스
Email: seojm@hanrw.ac.kr