한국시뮬레이션학회 논문지
Vol. 28, No. 3, pp. 1-9 (2019. 6)
http://doi.org/10.9709/JKSS.2019.28.3.001
ISSN 1225-5904

이산사건시스템 명세와 체계 요소 구조를 활용한 계층적 에이전트 합성 프레임워크

최창범 †

Hierarchical Agent Synthesis Framework using Discrete Event System Specification and System Entity Structure

Changbeom Choi†

ABSTRACT

An agent-based simulation is a popular simulation tool to solve various problems, such as stock market, population prediction, disease prediction, and development of a traffic system. As the agents are developed and researched in different application fields, the agent has a rigid structure and may not acceptable in different domains. As a result, it is a challenging problem to define a structure for an agent structure to reflect the researcher's simulation objective. This research proposes an extendable form for an agent and its modeling environment. In order to propose a standard structure, this study adopts system entity structure and discrete event system specification formalism. Also, this research introduces the SESManager which supports the proposed specification method. The proposed environment can hierarchically define the agent structure and synthesize the agent so that it can perform the agent simulation according to the user's simulation purpose.

Key words : Agent, Agent Modeling, Agent based Simulation, Simulation Tool

요 약

주식 시장, 인구 동향, 전염병 확산 예측, 도로교통체계 개발과 같이 다양한 분야에서 활용되고 있는 에이전트 기반 시뮬레이션 분야의 에이전트는 응용분야에 따라 각기 다른 형태로 발전되어왔다. 하지만 다학제 융합적 협력 요구되는 문제의 해결에 있어서 확장 가능한 형태의 에이전트 구조가 필요하며 이를 지원할 에이전트 모델링 및 시뮬레이션 환경이 필요하다. 본 연구는 다양한 분야의 에이전트를 수용하기 위하여 공통구조를 체계요소구조와 이산사건시스템형식론을 활용하여 명세할 수 있는 방법을 제시하고, 제시된 명세방법을 지원할 수 있는 개발 환경인 SESManager를 제안한다. 제안된 환경은 계층적으로 에이전트 구조를 정의하고 에이전트를 합성할 수 있도록 함으로써 사용자의 시뮬레이션 목적에 맞게 에이전트 시뮬레이션을 수행할 수 있도록 돕는다.

주요어 : 에이전트, 에이전트 모델링, 에이전트 기반 시뮬레이션, 시뮬레이션 도구

1. 서론

M&S 공학은 실세계 시스템을 다양한 수준으로 모사하여 컴퓨터 위에서 동작하는 시뮬레이션 모델을 만들고 이를 실행한 결과를 바탕으로 다양한 산업 제품의 설계와 제조 과정에서 최적화, 의사결정 등에 핵심적인 역할을 수행하고 있다. 이러한 M&S 공학 분야에서 빅데이터를 활용한 인공지능과 연계한 연구로 수행되고 있는 것이 에이전트 기반 모델링 및 시뮬레이션(Agent based

* 본 연구 논문은 한국전자통신연구원 연구운영비지원사업 (사물-사람-공간의 유기적 연결을 위한 초연결 공간의 분산 지능 핵심원천 기술, 19ZH1100) 위탁연구과제의 연구결과입니다.

Received: 27 April 2019, **Revised**: 25 July 2019, **Accepted**: 5 August 2019

† **Corresponding Author**: Changbeom Choi
E-mail: cbchoi@handong.edu
School of Global Entrepreneurship and ICT at Handong Global University

Modeling and Simulation: ABMS)이 있다. ABMS는 다른 M&S 공학에서 사용하고 있는 모델링과는 달리 전체 시스템을 구성하는 개체들에 대하여 에이전트로 모델링하고 에이전트와 에이전트 사이의 상호작용을 구현함으로써 시뮬레이션을 수행한다. 이와 같은 ABMS는 주식시장, 인구 동향, 전염병 확산 예측, 도로교통체계 개발 등에서 활용되고 있다(Bonabeau, 2002; Erol 등, 2007; Siddiqah, 2009). 이와 같은 ABMS는 시스템을 구성하는 에이전트를 M&S를 수행하는 연구자가 정의한 규칙을 바탕으로 환경 또는 다른 에이전트와 상호작용을 수행하고, 시뮬레이션이 수행되고 난 이후에 반복적으로 에이전트가 상호작용한 결과를 바탕으로 시뮬레이션 결과 분석을 수행한다.

다양한 분야에서 활용되고 있는 ABMS는 연구 분야에 따라 에이전트의 추상화 수준이 다르고 동작 방법도 다르므로 각 연구 분야에서 연구 의도에 맞게 에이전트 시뮬레이션 도구들이 개발되었다. ABMS 도구들은 기정의된 에이전트 구조를 활용하고 정의된 에이전트의 틀 안에서 파라미터를 변경하며 시뮬레이션을 수행한다. 따라서 기존의 ABMS 도구들의 에이전트들은 다른 분야의 시뮬레이션에서 그대로 활용하기에 불가능하며 새로운 종류의 에이전트를 추가하기에도 많은 제약사항이 따른다. 더욱이 다학제 융합 연구에 있어 다양한 종류의 에이전트들을 연구자의 의도에 따라 구성하고 에이전트 간의 상호작용에 참여시켜 동작시켜야 하므로 에이전트의 구조를 쉽게 변경하고 새로운 종류의 에이전트를 생성할 수 있도록 만드는 구조 정립이 필요하다.

본 연구는 ABMS에서 시뮬레이션을 수행하는 연구자의 의도를 효과적으로 반영하기 위하여 시뮬레이션의 목적에 부합하는 에이전트를 합성할 수 있도록 계층적인 에이전트의 구조 및 프레임워크를 제안한다. 제안하는 계층적 프레임워크는 체계요소구조(System Entity Structure: SES)를 활용하여 에이전트가 갖추어야 할 기본적인 요소를 정의하고 이를 확장하는 과정에서 다양한 대안을 갖출 수 있도록 에이전트의 구조를 정의한다. 또한, 실제 에이전트 기반 시뮬레이션에서 시뮬레이션 목적에 부합되는 에이전트를 합성하는 시스템을 제공한다.

2. 연구배경 및 관련연구

본 장에서는 계층적 에이전트 합성 프레임워크를 제안하기에 앞서 에이전트의 정의와 구조를 소개하고, 에이전트 합성을 위한 관련 연구를 소개한다.

2.1 에이전트 기반 모델링 및 시뮬레이션

에이전트란 환경에서 발생하는 사건들을 인식하고 처리하여 적절한 동작을 발생시켜 환경에 영향을 주는 동작을 수행하는 모든 것으로 정의할 수 있다(Russell 등 2016). 이와 같은 정의를 바탕으로 에이전트는 다양하게 구성할 수 있으며 소프트웨어뿐만 아니라 하드웨어 등으로도 에이전트의 개념을 구현할 수 있다.

에이전트의 핵심구조로는 환경으로부터 사건을 인식할 수 있는 인식기(Sensor)와 인식된 사건들을 바탕으로 동작을 결정하는 결정기(Decision), 동작을 수행하는 동작기(Actuator)가 있다. 에이전트의 정의에 따라 에이전트는 부착된 인식기를 통해서만 환경에서 발생하는 사건을 인지할 수 있으며 입력 사건들에 대하여 처리할 수도 있다. 동작기도 인식기와 유사하게 에이전트에 내재된 동작기를 통해서만 환경에 영향을 미칠 수 있으며 다수의 동작을 수행할 수 있다. 따라서 에이전트를 정의하면서 결정되는 인식기와 동작기보다 환경으로부터 발생하여 에이전트가 인지한 사건들에 대하여 어떤 동작을 발생시켜 환경에 영향을 줄 것인지를 결정하는 결정기에 관한 연구가 활발히 진행되었다(Hwang 등 2009).

2.2 이산사건시스템 명세

이산사건시스템(Discrete Event System: DEVS) 명세는 DEVS 형식론을 바탕으로 시스템을 수학적 틀로 표현하는 명세 기법이다(Zeigler 등, 2000). DEVS 명세는 시스템을 표현할 때 모델 명세를 명확하게 정의하기 위하여 수학적으로 모호함이 없는 형태로 시스템의 동작을 기술하는 방법으로 다수의 사용자가 DEVS 명세를 보았을 때 같은 정보와 이해를 제공할 수 있도록 돕는 방법이다. DEVS 형식론은 시스템을 모델링할 때 더 나눌 수 없는 단위 요소로 나누고 이를 원자 모델(Atomic 모델)이라 정의한다. 다음은 원자 모델에 대한 명세 방법이다.

$$AM = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X : Input Events Set

Y : Output Events Set

S : States Set

$\delta_{ext} : Q \times X \rightarrow S$; External Transition Function

$Q = \{(s,e) | \ s \in S, \ 0 \le e \le ta(s) \}$; Total state of AM,

e : elapsed time

$\delta_{int} : S \rightarrow S$; Internal Transition Function

$\lambda : S \rightarrow Y$; Output Function

$ta : S \rightarrow R_{\infty}^{+}$; Time Advance Function

원자 모델은 외부로부터 입력 사건이 입력되었을 때의 상태와 입력사건의 종류에 따라 시스템이 어떤 동작을 수행하는 지 외부천이함수(External Transition Function)를 통해서 명세하고, 시스템에 특정 시간 동안 외부 입력이 없을 때의 동작을 내부천이함수(Internal Transition Function)으로 명세한다. 원자모델은 외부 입력이 없어 내부천이 함수가 호출되는 시점에서 출력함수(Output Function)에 따라 외부 사건을 발생시킬 수 있으며 시간 전진 함수를 사용하여 시스템의 각 상태에 따른 내부천이함수 호출 시점을 결정한다.

또한, DEVS 명세는 결합 모델(Coupled Model)을 활용하여 모델의 구조적 특징과 계층적인 특징을 표현한다. 다음은 결합모델의 정의이다.

$$CM = < X, Y, \{M_i\}, EIC, EOC, IC, SELECT >$$

X : Input Events Set

Y : Output Events Set

$\{M_i\}$: Component Models Set

$EIC \subseteq X \times \bigcup_i X_i$: External Input Coupling Relation

$EOC \subseteq \bigcup_i Y_i \times Y$: External Output Coupling Relation

$IC \subseteq \bigcup_i Y_i \times \bigcup_j X_j$: Internal Coupling Relation

$SELECT : 2^{\{M_i\}} - \phi \rightarrow M_i$

여러 모델을 내부적으로 연결하여 만든 모델로, 입력 집합, 출력집합과 모델 집합을 이용하여 외부의 시스템들과 연결되는 인터페이스를 정의한다. 인터페이스는 각각 외부에서 입력된 사건을 내부의 하위시스템에 전달할 수 있는 연결 관계(External Input Coupling)와 내부에서 외부로 연결되는 외부출력연결(External Output Coupling) 관계, 내부에서 내부로 연결되는 내부연결(Internal Coupling) 관계로 명세 된다. 이때 내부의 구성요소는 원자 모델 혹은 결합모델로 표기될 수 있으며 사건이 발생 시에 관계로 연결된 모든 하위시스템이 해당 사건을 처리한다. 이상과 같은 결합모델은 그 구조를 Fig. 1과 같이 도식화하여 표현할 수 있다.

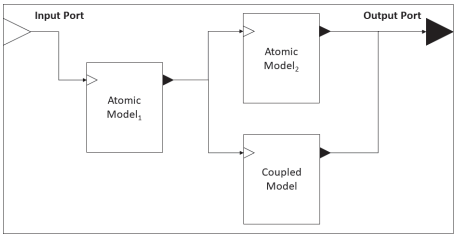

Fig. 1. Diagram of Coupled Model

원자모델과 결합모델은 계층적 에이전트의 기본 구성요소를 표현하기에 충분한 표현력을 가지고 있다. 특별히 원자 모델은 에이전트의 상태에 따른 동작에 대하여 표현할 수 있으며 사건을 발생시키는 발생기, 사건을 받아 처리하는 수용기, 사건을 저장 분배하는 분배기의 형태로 에이전트의 기본구성요소를 표현할 수 있으며 이를 결합모델을 사용하여 각 구성요소를 조합하여 에이전트를 구성할 수 있다.

2.3 체계요소구조

체계요소구조(System Entity Structure: SES)는 시스템을 표현하는 방식에 대해서 다측면적으로 시스템을 이해하고 이를 바탕으로 체계가 가질 수 있는 모든 대안을 표현하는 방식이다. SES는 시스템의 다양한 대안을 총체적으로 표현하기 위해서 시스템을 구성하고 있는 하위시스템의 구성 관계, 연결 관계, 분류 관계를 명확하게 기술하기 위한 방법론이다(Zeigler, 1991).

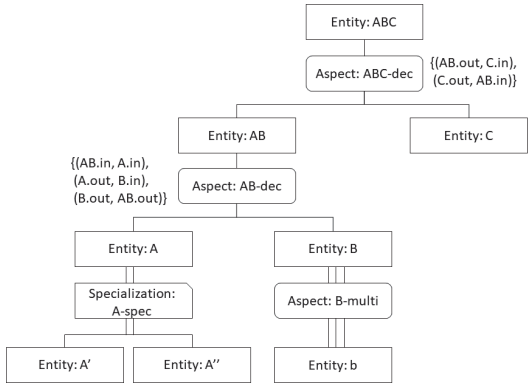

Fig. 2. Example of System Entity Structure

Fig. 2에서 직각 사각형으로 도시한 ABC, AB, C, A, B, A′, A″, b는 Entity 노드로 실제 시스템에 대응된다. 모서리가 둥근 사각형인 ABC-dec, AB-dec, A-spec, B-multi는 Aspect 노드로 하나의 Entity 노드가 어떤 Entity 노드들로 구성되어 있는지, 이들 Entity 노드 사이에 어떤 관계를 구성하는지 나타낸다. 특별히, 시스템을 구성하는 Entity가 복수 개를 담을 수 있는 경우에는 Multi-Aspect를 기술할 수 있으며 몇 개의 Entity로 구성되는지를 기록할 수 있는 변수의 개념을 담고 있다. 마지막으로 한쪽 모서리가 잘린 사각형으로 표현되는 A-spec은 Specialization 노드이며 시스템의 Entity가 가질 수 있는 대안들을 나타내고 두 줄로 표현한다. Fig. 2에서는

A가 A′, A″의 대안을 가질 수 있으며 시스템을 구성하는 과정에서 주어진 기준에 따라 A′, A″를 선택하여 시스템을 구성할 수 있다.

이상의 SES 명세를 활용하여 에이전트의 구조를 정의하면 계층적으로 에이전트의 모델을 기술할 수 있으며 에이전트의 구성요소를 객체 지향적으로 표현하고 이를 조합하는 형태로 다양한 에이전트를 합성할 수 있다. 특별히 SES의 Entity 노드를 통하여 에이전트 기반 시뮬레이션을 실행하기 위해서 에이전트가 반드시 갖추어야 하는 인터페이스 요구사항 및 동작에 대한 요구사항을 명세할 수 있으며 Specialization 노드를 사용하여 다양한 대안에 대하여 명세하고 이를 에이전트 기반 시뮬레이션을 구성하는데 활용할 수 있다. 또한, 시뮬레이션을 구성하는데 Entity에 해당하는 모델을 저장/관리하는 모델 데이터베이스를 활용할 수 있다.

2.4 관련 연구

일반적인 ABMS에서 에이전트에 대한 연구로 에이전트의 동작을 명세하는 방법과 구조를 변화시키는 방안에 대한 연구로 나눌 수 있다. 일반적으로 에이전트의 동작을 정의하기 위하여 에이전트의 상태와 에이전트의 동작규칙을 정의하고 규칙에 대한 파라미터를 조정해가며 시뮬레이션을 수행한다(Macal 등 2005; Railsback 등 2006). 이러한 에이전트는 시뮬레이션 목적이 변경되었을 때 파리미터로 에이전트의 동작을 묘사할 수 없는 경우 기존 에이전트를 활용할 수 없는 문제가 있다.

이외에 에이전트를 하나의 시뮬레이션 모델로 정의하고 에이전트의 구성요소를 재조합하고 사용할 수 있는 연구로 Java 기반의 MS4Me를 사용한 시뮬레이션 환경이 제시되었다(Zeigler 등 2013; Zeigler 등 2017). MS4Me 환경의 ABMS는 동적으로 구조를 변화시키는 연구로 에이전트 기반 시뮬레이션을 수행하는 도중에 에이전트를 추가하는 방법에 대한 연구가 진행되었다. 이를 위하여 MS4Me는 Publish/Subscribe 방식의 에이전트 관리 방법을 채용하여 새로운 에이전트가 시뮬레이션에 참여하는 과정에서 에이전트들에서 생성하는 메시지와 구독하는 메시지를 정의하고, 이들 메시지를 관리하는 PublishSubscribeRouter 모델과의 프로토콜을 정의하고 이 프로토콜이 구현된 모델에 대해서 에이전트 시뮬레이션을 지원한다. 제안하는 에이전트 명세 및 합성 프레임워크는 SES와 SES로부터 시뮬레이션을 합성하는 방법에 있어서 MS4Me와 유사하나 ABMS를 수행하고자 하는 연구자들을 돕기 위하여 에이전트의 요구사항을 정립하고 그 구조를 계층적으로 정의하며 다른 연구자들이 에이전트를 기계학습들에 활용할 수 있도록 확장하고 데이터베이스에 저장하여 재사용할 수 있도록 지원하는 환경이다.

3. 계층적 에이전트 명세 및 합성 프레임워크

에이전트 기반 시뮬레이션에서 활용할 에이전트를 정의하기 위해서는 에이전트의 구성요소에 대한 요구사항 정립이 필요하며, 독립성, 완전성, 상호호환성에 대한 요구사항을 정리할 수 있다.

계층적 에이전트 구성요소에 대한 요구사항으로 독립성은 구성요소가 다른 개체의 구성요소로 활용될 수 있는 요구사항이다. 이를 위하여 에이전트의 구성요소는 타 구성요소와 상호작용할 수 있는 인터페이스가 확립되어 있어야 하며 구성요소의 동작을 결정지을 수 있는 명세 방법이 필요하다. 에이전트 구성요소의 완전성은 시뮬레이션 목적에 부합되는 에이전트의 구성요소를 검색하여 구성할 때 에이전트와 시뮬레이션 실행 환경에서 대응되는 시뮬레이션 모델이 존재해야 함을 의미한다. 상호호환성 요구사항은 시뮬레이션 목적에 부합하는 구성요소들을 찾아 에이전트를 합성할 때 시뮬레이션 목적에 부합하는 결과를 도출하는 데 필요한 요구사항이다. 이상과 같은 요구사항을 바탕으로 계층적 에이전트의 각 구성요소에 대한 구조 명세를 소개한다.

본 연구의 합성 가능한 계층적 에이전트는 DEVS 명세와 SES를 활용하여 에이전트의 구조 요구사항에 부합하는 구성요소를 설계한다. 특별히 DEVS 명세는 DEVS 형식론의 원자모델과 결합모델의 집합론 기반의 조립이 가능한 형식론이라는 특징을 활용할 수 있어 합성될 에이전트가 가져야 하는 요구사항에 부합한다. 다음 Fig. 3는 계층적 에이전트의 센서부이다.

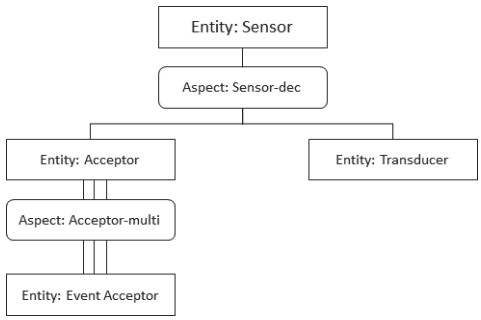

Fig. 3. Structure of Sensor Component for an Agent

에이전트의 센서기는 환경과 에이전트로부터 다양한 종류의 이벤트를 전달받을 수 있으므로 다양한 종류의 이벤트를 수용할 수 있는 형태로 Multi-aspect로 Acceptor Entity를 표현하고 다수의 Event Acceptor를 구성할 수 있다. 이벤트 변환기는 에이전트의 처리기에서 처리할 수 있는 형태로 이벤트를 가공하고 이를 에이전트의 처리기로 전달하는 역할을 담당한다. 다음 Fig. 4는 Fig. 3의 구조를 결합모델 명세로 표현한 것이다.

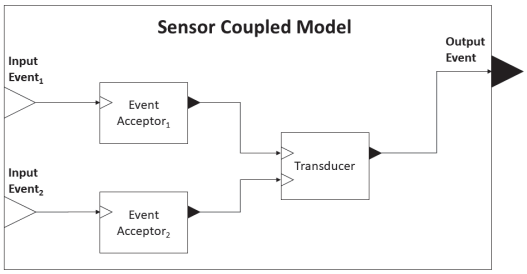

Fig. 4. Coupled Model of Sensor Component

계층적 에이전트의 처리기는 센서기로부터 받은 이벤트를 받아 저장하다 일괄처리하는 Acceptor Entity, 외부에서 전달받은 이벤트를 처리하는 Decision Entity와 외부에 연결된 구동기에 호환 가능한 이벤트를 생성하는 Transducer Entity로 구성된다. Decision Entity의 경우 저장하여 에이전트의 상태를 표현하는 History Management Entity와 에이전트 외부로 동작을 생성하는 Event Generator Entity로 구성할 수 있다. 특히, 외부에서 발생한 이벤트를 저장하여 기계학습을 이용하기 위해서는 History Management Entity와 Event Generator Entity를 연계하여 표현할 수 있다. 다음은 Fig. 5은 Agent의 Processor Entity의 시스템 구조이다.

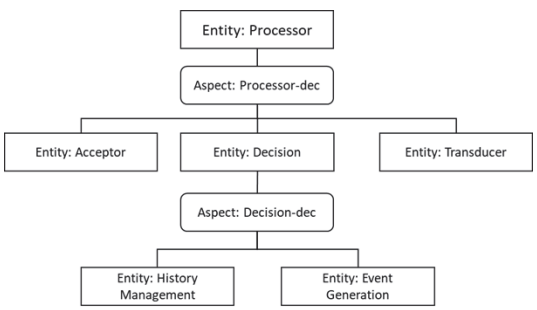

Fig. 5. Structure of Processor Component for an Agent

만일 Processor Entity의 History Management Entity를 확장하기 위해서 SES 구조의 Specialization Attribute를 활용하여 SES를 확장하고 이를 Agent Structure Database에 저장할 수 있다. Fig. 6은 History Management Entity에 대하여 아무런 동작을 수행하지 않는 Entity인 Null Manager Entity와 기계학습을 적용한 History Manager을 나타내는 ML Manager Entity로 확장한 것이다.

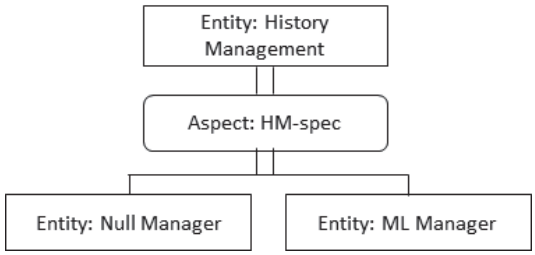

Fig. 6. Example of Specialization for History Management Entity

계층적 에이전트의 구동기는 에이전트의 처리기에서 생성된 에이전트의 동작 이벤트에 따라 동작 생성을 담당하는 구성요소이다. 구동기는 에이전트의 다른 구성요소와 비슷하게 생성된 이벤트를 받아 저장하고 이를 처리기로 전달하는 Acceptor Entity와 실제 이벤트를 생성시켜 에이전트 외부로 전달하는 Generator Entity가 있다. 다음 Fig. 7은 Actuator Entity에 대한 SES이다.

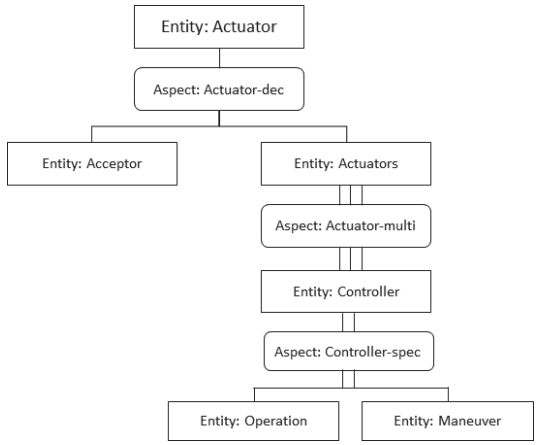

Fig. 7. Structure of Actuator Component for an Agent

이상의 이산사건시스템 형식론과 SES를 활용한 계층적 에이전트 명세를 바탕으로 한 에이전트 합성 프레임워크는 크게 3가지로 구성된다. 사용자의 요구사항을 받아 이를 처리하는 구조 정리기(Structure Pruner), 정리된

구조를 바탕으로 DEVS 모델 데이터베이스로부터 획득된 대안들을 선택하는 대안 선택기(Alternative Selector)와 정리된 구조대로 선택된 구성요소들을 합성하는 에이전트 합성기(Agent Synthesizer)로 구성되어 있다. Fig. 8는 계층적 에이전트 합성 프레임워크를 도식화한 것이다.

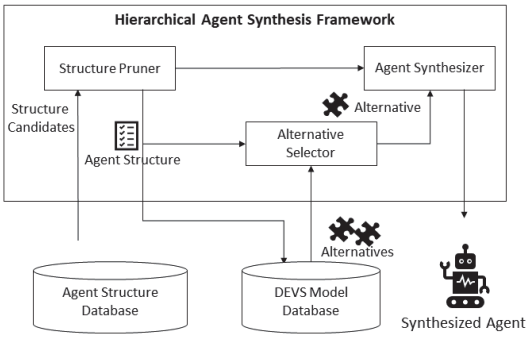

Fig. 8. Proposed Hierarchical Agent Synthesis Framework

본 연구에서는 에이전트 합성 프레임워크에서 활용하기 SES를 처리하는 프로그래밍 환경인 SESManager을 구축하였다(Choi, 2019). SESManager은 사용자와 상호작용하여 에이전트의 SES를 생성하고 관리할 수 있다. SES의 Entity와 Attribute 노드를 표현하기 위해서 SESManager에서는 Entity의 이름과 Agent의 이름을 표현하는 'name' 키를 사용하며, Entity 이후에 Attribute 노드가 반드시 따라와야 하는 SES 방법론의 정의대로 core_attribute를 두어 해당 entity를 표현할 수 있다. Table 1은 core_attribute의 구성요소에 대한 설명이다.

Table 1. Description of System Entity Structure for an Agent

Key	Item	Description
entities	[entity, arity, optional]	sub-components of the given entity
external_input	port:[entity, port]	external interface for input events
external_output	port:[entity, port]	external interface for output events
input_ports	[$port_1$, ..., $port_n$]	types of an input events
output_ports	[$port_1$, ..., $port_n$]	types of an output events
internal	entity:{port:[entity, port]}	interface among sub-components
type	string	type of an entity

core_attribute는 기본적으로 DEVS 형식론의 결합모델로 표현된 모델을 표현하기 위하여 external_input, external_output, input_port, output_ports, internal에 대한 항목을 저장한다. 이외에 core_attribute에서 기존의 DEVS 형식론과는 달리 확장된 부분은 entities로 기존 결합모델을 표현하는 방식에서는 해당 모델을 구성하는 다른 모델들로 구성된 집합으로 표현되었다면 계층적 에이전트 합성프레임워크에서는 에이전트를 합성하는 데 필요한 정보로 다수 개의 Entity로 구성되는지, 필수적으로 구성되는지를 표현하는 항목으로 구성된다.

4. 사례 연구

제안하는 계층적 에이전트 명세 및 합성 프레임워크의 정확성 검증과 응용가능성을 보이기 위하여 기존에 잘 알려져 있는 C++기반의 DEVS 형식론 시뮬레이션 환경인 DEVSim++을 활용하여 검증을 수행하였다(Young 등 1993). DEVSim++는 C++ 프로그래밍 언어의 특징인 객체지향성을 지원하는 시뮬레이션 환경으로 DEVS 형식론의 원자모델과 결합모델에 해당되는 C++ 클래스를 구현하면 DEVS형식론의 시뮬레이션 알고리즘대로 해당 클래스 인스턴스를 실행하는 시뮬레이션 환경이다. DEVSim++는 국방 M&S 등에 최적화되어 다양한 국방 워게임 모델에 활용되었기 때문에 제안하는 DEVSim++의 시뮬레이션 모델과 동일한 의미를 가지도록 제안하는 환경에 이식하고 제안하는 합성 알고리즘에 적용하여 검증을 수행하다. 검증에 활용된 시뮬레이션 모델은 DEVSim++의 BankSim으로 Generator, Queue Teller로 구성된 모델이다. 다음 Fig. 9는 검증에 사용된 시뮬레이션 모델의 구성도이다.

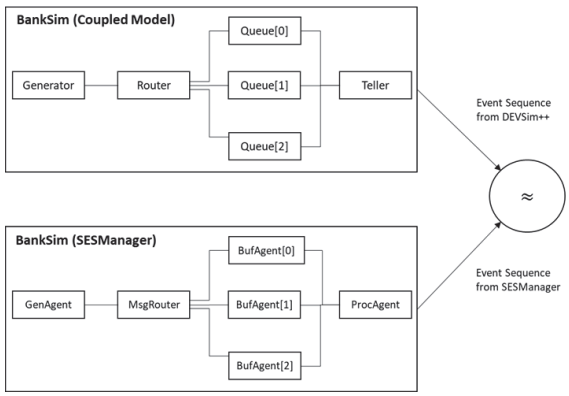

Fig. 9. Structure of BankSim Simulators

검증에 사용된 DEVSim++으로 구성된 시뮬레이션 모델은 결합모델에서 원자모델을 구성요소로 사용하여 시뮬레이션 모델의 구조를 결정한다. 이와 달리 SESManager로 구성된 시뮬레이션은 결합모델이 없이 원자모델로 구성된 시뮬레이션 모델을 시뮬레이션 실행시간에 조합하여 시뮬레이션을 수행한다. 다음 Fig. 10은 제안된 시뮬레이션 환경에서 생성된 시뮬레이션 모델의 Pruned Structure이다.

```
{
    "name": "BankSim",
    "core_attribute": {
        "entities": [
            ["GenAgent", 1, false],
            ["MsgRouter", 1, false],
            ["BufAgent", 3, false],
            ["ProcAgent", 1, false]
            ],
            "input_ports": [],
            "output_ports": [],
            "external_input": {},
            "external_output": {},
            "internal": {
              "GenAgent": [
                  ["task",["MsgRouter","task_in"]]
               ],
               "MsgRouter": [
                  ["task_out[0]", ["BufAgent[0]","task_in"]],
                  ["task_out[1]", ["BufAgent[1]","task_in"]],
                   ["task_out[2]", ["BufAgent[2]","task_in"]]
               ],
              "BufAgent[0]": [
                  ["task_out",["ProcAgent","task_in"]]
              ],
              "BufAgent[1]": [
                  ["task_out",["ProcAgent","task_in"]]
              ],
              "BufAgent[2]": [
                  ["task_out",["ProcAgent","task_in"]]
              ]
            }
    },
    "optional_attributes": {}
}
```

Fig. 10. Pruned Structure of BankSim in SESManager

다음 Fig. 11과 Fig. 12는 제안된 시뮬레이션 환경의 검증을 위하여 사용된 DEVSim++ 모델의 원자모델 소스코드와 SESManager의 소스코드이다.

```
class Gen: public CAtomic
{
public:
    Gen(int _item = 100)
    {
        SetName("GenAgent");

        m_state = GEN;
        cur_item = 0;
        gen_item = _item;
        AddOutPorts(1, "task");
    }

public:
    bool OutputFn(CDEVSimMessage& msg) override
    {
        Message* contents = new Message(cur_item++,
                                        "gen");
        msg.SetPortValue("task", contents, false);
        return true;
    }

    bool IntTransFn() override
    {
        if (cur_item >= gen_item)
            m_state = WAIT;
        return true;
    }

    TimeType TimeAdvanceFn() override
    {
        if (m_state == GEN)
            return 1;
        else
            return Infinity;
    }
private:
    enum State{ GEN, WAIT};
    State m_state;
    int cur_item;
    int gen_item;
};
```

Fig. 11. Generator Model in DEVSim++

Gen 원자모델과 GenAgent 모델을 포함한 모든 시뮬레이션 모델 소스코드는 동일한 의미를 가질 수 있도록 이식되었고, 결합모델 대신 앞서 제시한 Pruned Structure로 합성하였으며 다수의 Buf모델을 구축하였을 시에 두 시뮬레이션 모두 동일한 시뮬레이션 결과를 도출함을 확인하였다. 다음 Fig. 13과 Fig 14는 DEVSim++로 실행된 시뮬레이션 결과와 SESManager에서 획득한 시뮬레이션 결과의 일부를 발췌한 것이다.

```
class GenAgent(BehaviorModelExecutor):
    def __init__(self, instance_time,
          destruct_time, name, engine_name, _item):
        BehaviorModelExecutor.__init__(self,
          instance_time, destruct_time,
          name, engine_name)

        self.init_state("GEN")
        self.insert_state("GEN", 1)
        self.insert_state("WAIT", "inf")

        self.insert_output_port("task")

        self.cur_item = 0
        self.gen_item = _item

    def ext_trans(self,port, msg):
        pass

    def output(self):
        msg = SysMessage(self.get_name(), "task")
        msg.insert(self.cur_item)
        msg.insert("gen")
        self.cur_item = self.cur_item + 1

        return msg

    def int_trans(self):
        if self.cur_item > self.gen_item:
            self._cur_state = "WAIT"

    @staticmethod
    def duplicate(instance_time,
            destruct_time, name, engine_name, _item):
        return Gen(instance_time, destruct_time,
                   name, engine_name, _item)
```

Fig. 12. GenAgent Model in SESManager

```
2019.07.04 17:42:22.985721 | 6 [DEVSim++ Simulation Engine] SIMULATION MODE
2019.07.04 17:42:22.988722 | 25 MSG_ID:0, Path:gen->BufAgent[1]
2019.07.04 17:42:23.010722 | 41 MSG_ID:1, Path:gen->BufAgent[2]
2019.07.04 17:42:23.013720 | 73 MSG_ID:4, Path:gen->BufAgent[0]
2019.07.04 17:42:23.014721 | 89 MSG_ID:5, Path:gen->BufAgent[1]
2019.07.04 17:42:23.016721 | 105 MSG_ID:6, Path:gen->BufAgent[2]
2019.07.04 17:42:23.019724 | 137 MSG_ID:9, Path:gen->BufAgent[0]
2019.07.04 17:42:23.025722 | 153 MSG_ID:10, Path:gen->BufAgent[1]
2019.07.04 17:42:23.040724 | 169 MSG_ID:11, Path:gen->BufAgent[2]
2019.07.04 17:42:23.047722 | 201 MSG_ID:14, Path:gen->BufAgent[0]
2019.07.04 17:42:23.051736 | 217 MSG_ID:15, Path:gen->BufAgent[1]
```

Fig. 13. Simulation Results of DEVSim++

```
>>> runfile('D:/Projects/SESManager/sample/
MSG_ID:0, Path: gen->Buf[1]
MSG_ID:1, Path: gen->Buf[2]
MSG_ID:4, Path: gen->Buf[0]
MSG_ID:5, Path: gen->Buf[1]
MSG_ID:6, Path: gen->Buf[2]
MSG_ID:9, Path: gen->Buf[0]
MSG_ID:10, Path: gen->Buf[1]
MSG_ID:11, Path: gen->Buf[2]
MSG_ID:14, Path: gen->Buf[0]
MSG_ID:15, Path: gen->Buf[1]
```

Fig. 14. Simulation Results of SESManager

5. 결론

다학제 융합 연구에서 활발히 활용되고 있는 에이전트 기반 시뮬레이션은 연구자의 의도에 맞게 에이전트를 정의하고, 에이전트를 구현하여 시뮬레이션을 수행하는 것이 중요하다. 일반적으로 에이전트 기반 시뮬레이션의 경우 에이전트의 구조를 정형화하여 시뮬레이션 목적에 맞게 파라미터를 변경하며 시뮬레이션을 수행하였다. 하지만, 시뮬레이션의 수행 목적의 변화에 따라 다양한 종류의 에이전트를 시뮬레이션 목적에 맞게 구성하여 빠르게 시뮬레이션에 적용하거나 기계학습과 같은 인공지능 기술이 가미된 에이전트 확장 측면에서는 제한점이 많이 있을 수 있다.

본 논문에서는 ABMS 분야에서 에이전트의 재사용성을 높이고 인공지능과 같은 타학문 분야와의 융합을 수월하게 하기 위하여 에이전트의 구조를 정의하고 이를 시뮬레이션할 수 있는 방법으로 SES와 이에 대응하는 DEVS 형식론 기반 모델을 활용하여 에이전트의 구조를 계층적으로 명세하는 방법과 이를 구현한 SESManager를 소개하였다. 제안하는 에이전트 구조는 외부 이벤트를 받아 내부적으로 처리할 수 있는 형태로 변환하는 센서기, 이벤트를주기적으로 처리하고 동작을 생성하는 처리기, 에이전트의 동작을 생성하는 구동기로 정의하고 각 센서기, 처리기, 구동기 내에 기계학습 알고리즘과 같이 타 학문분야의 성과를 결합시키기 위한 구조를 제안하였다. 그리고 에이전트의 구조를 기반으로 에이전트를 관리하고 생성시키는 실행환경인 SESManager를 구현하였다. 제안하는 환경을 통해 ABMS를 활용하는 연구자들이 개발된 에이전트를 재활용하고 빠르게 에이전트 기반 시뮬레이션을 수행할 수 있고 인공지능 분야와의 융합을 수월하게 진행할 수 있을 것으로 기대된다. 향후 연구로는 SES와 대응되는 시뮬레이션 모델에 대한 검증 및 성능향상에 관한 연구가 필요하다.

References

Bonabeau, E. (2002), Agent-based modeling: Methods and techniques for simulating human systems Proceedings of the National Academy of Sciences. May 14, 2002

Choi, Changbeom (2019). SESManager: System Entity Structure Manager (Version 1.0) [Software]. Available from https://github.com/cbchoi/SESManager

Erol, K., Levy, R., and Wentworth, J. (2007) Application of Agent Technology to Traffic Simulation, United States Department of Transportation, May 15, 2007

Hwang, M. et. al. (2009) Rule-Based Simulation of Multi-Cellular Biological Systems－A Review of Modeling Techniques: Cellular and Molecular Bioengineering, Vol. 2, No. 3, pp. 285-294, 2009

Macal, C. M., and Michael J. N. (2005) Tutorial on agent-based modeling and simulation, Proc. of the 37th conference on Winter simulation, Winter Simulation Conference, 2005.

Railsback, S. F., Steven L. L., and Stephen K. J. (2006), Agent-based simulation platforms: Review and development recommendations, Simulation 82.9, 609-623, 2006

Russell, S. J., and Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,.

Siddiqah A. et. al. (2009). A new hybrid agent-based modeling decision support system for breast cancer research, IEEE ICICT, IBA, Karachi, August 15-16, 2009. Breast Cancer DSS

Young C. Kim, Kyung S. Ham, and Tag Gon Kim (1993). Object-Oriented Memory Management in DEVSim++, Proceedings of the 1993 Winter Simulation Conference, 1993.

Zeigler, B. P., Luh, C.J., and Kim, T.G. (1991). Model Base Management for Multifacetted Systems, ACM Transactions on Modeling and Computer Simulation - TOMACS, Vol. 1, No. 3, pp. 195-218, 1991.

Zeigler, B. P., Praehofer, H. and Kim, T.G. (2000). Theory of Modelling and Simulation (2nd Edition), Academic Press, 2000.

Zeigler, B. P., Chung, Kim Do (2013). System Entity Structures for Suites for Simulation Models, International Journal of Modeling, Simulation, and Scientific Computing 4(3), 2013.

Zeigler, B.P., Sarjoughian, H. S. (2017). Guide to Modeling and Simulation of Systems of Systems (2nd Edition), Springer International Publishing, 2017.

최 창 범 (cbchoi@handong.edu)

2005 경희대학교 컴퓨터공학 학사
2007 KAIST 전산학 석사
2014 KAIST 전자공학 박사
2014~ 현재 한동대학교 ICT창업학부 조교수

관심분야 : DEVS 형식론, 소프트웨어 품질 보증, VV/A