JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 27 No. 5, pp. 101-107, May 2022
https://doi.org/10.9708/jksci.2022.27.05.101

Script-based Test System for Rapid Verification of Atomic Models in Discrete Event System Specification Simulation

Su-Man Nam*

*Researcher, DuDu Information Technologies, Ltd., Seoul, Korea

[Abstract]

Modeling and simulation is a technique used for operational verification, performance analysis, operational optimization, and prediction of target systems. Discrete Event System Specification (DEVS) of this representative technology defines models with a strict formalism and stratifies the structures between the models. When the atomic DEVS models operate with an intention different the target system, the simulation may lead to erroneous decision-making. However, most DEVS systems have the exclusion of the model test or provision of the manual test, so developers spend a lot of time verifying the atomic models. In this paper, we propose a script-based automated test system for accurate and fast validation of atomic models in Python-based DEVS. The proposed system uses both the existing method of manual testing and the new method of the script-based testing. As Experimental results in our system, the script-based test method was executed within 24 millisecond when the script was executed 10 times consecutively. Thus, the proposed system guarantees a fast verification time of the atomic models in our script-based test and improves the reusability of the test script.

▸ **Key words**: Modeling and Simulation, Discrete Event System Specification, Model Test, Script-based Test, Verification and Validation

[요 약]

모델링 및 시뮬레이션은 목표 시스템의 동작 검증, 성능 분석, 운용 최적화, 예측을 위해 사용되는 기술이다. 이 기술의 대표적인 이산사건 시스템 명세(DEVS)는 모델들을 엄격한 형식론으로 정의하고 모델 간의 구조를 계층화한다. 이 DEVS 모델들의 원자 모델은 목표와 다른 의도로 동작하게 될 경우 시뮬레이션은 잘못된 의사결정으로 이어질 수 있다. 그럼에도 대부분 DEVS 시스템은 모델 테스트의 부재 또는 수동 테스트 환경으로 제공하여 개발자가 모델을 검증하는 데 오랜 시간이 소비된다. 본 논문에서는 파이썬 기반 DEVS에서 정확하고 빠른 원자 모델의 검증을 위해 스크립트 기반 테스트 시스템을 제안한다. 제안 테스트 시스템은 기존 방식인 수동 테스트와 새로운 방식인 스크립트 기반 테스트를 둘 다 사용한다. 우리 시스템의 실험 결과, 제안 테스트 방식은 스크립트를 10번 연속 실행 시 24ms 이내에 실행되었다. 그리하여 제안 시스템은 스크립트 기반 테스트를 사용해서 빠른 원자 모델 검증 시간을 보장하고, 테스트 스크립트의 재사용성을 향상한다.

▸ **주제어**: 모델링과 시뮬레이션, 이산사건시스템 명세, 모델 테스트, 스크립트 기반 테스트, 검증과 확인

• First Author: Su-Man Nam, Corresponding Author: Su-Man Nam
*Su-Man Nam (sumannam@gmail.com), DuDu Information Technologies, Ltd.
• Received: 2022. 03. 17, Revised: 2022. 04. 19, Accepted: 2022. 04. 25.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

모델링 및 시뮬레이션은 수집된 데이터를 기반으로 시스템의 검증, 성능 분석, 예측을 위해 다양한 분야(스마트 제조, 워게임 등)에서 응용되고 있다[1, 2]. 이 시뮬레이션 방법론은 크게 연속적인 시뮬레이션과 이산 사건 시뮬레이션으로 구분된다. 연속적인 시뮬레이션은 실세계에 가깝고 정확함에도 이산 사건 시뮬레이션보다 판단해야 하는 정보량과 시간 낭비가 크다. 반면에 이산 사건 시뮬레이션은 중요한 사건을 중심으로 시뮬레이션하여 연속 시뮬레이션 대비 처리 시간이 빠르다.

Zeigler에 의해 개발된 이산사건 시스템 명세(Discrete Event System Specification; 이하 DEVS)[3-5]는 이산 사건 이벤트를 추상화하여 복잡한 시스템을 모델링하는데 널리 사용하는 형식론이다. DEVS의 장점은 모델을 엄격한 형식론으로 정의하고 모델 간의 구조를 계층적으로 표현한다. DEVS는 시스템의 구조를 표현하는 결합 모델(coupled model)과 행위(behavior)를 표현하는 원자 모델(atomic model)로 구성된다. 그중 원자 모델은 시스템의 행위를 세 가지 집합과 네 가지 함수로 표현한다. 그리하여 원자 모델은 시뮬레이션의 올바른 동작을 위해 집합들과 함수들의 결과를 절차적으로 검증해야만 한다.

DEVS 기반의 시뮬레이션 시스템은 다양하게 연구되고 있는데, 대표적으로 DEVS-Scheme[1, 6, 7], DEVSJava[8, 9], DEVS-Python[10], ADEVS[11], DEVSim++[12] 등이 있다. 그중 원자 모델의 정확한 검증을 위한 테스트 환경은 DEVS-Scheme, DEVS-Python 등 소수의 시스템에서만 제공하고 있다.

특히, DEVS-Python은 Python을 사용하여 쉽게 모델링하고 시뮬레이션할 수 있을 뿐만 아니라 원자 모델 테스트 환경도 제공한다. 그럼에도 원자 모델 테스트 환경이 콘솔(console) 창에서 직접 타이핑(typing)하므로 모델 검증 시간이 개발자의 타이핑 속도에 영향을 받는다. 따라서 개발자가 원자 모델의 빠른 테스트를 위해 테스트 절차를 자동화할 경우 DEVS 모델의 검증 시간을 줄일 뿐만 아니라 재사용성도 향상할 수 있다.

본 논문에서는 이산 사건 시뮬레이션인 DEVS-Python에서 모델 검증 시간을 줄이기 위해 스크립트(script) 기반 테스트 시스템을 제안한다. 제안 시스템은 원자 모델을 위한 스크립트 재사용성을 제공하고 이를 통해 빠른 테스트 시간을 보장한다. 이에 따라 개발자는 간편하고 빠르고 정확하게 DEVS 원자 모델을 구현할 수 있다. 본 시스템의 실험 결과, 스크립트 기반 원자 모델 테스트 사용으로 4ms의 실행 시간을 보장한다. 본 논문의 기여는 다음과 같다.

- 스크립트 기반 원자 모델 테스트 환경 제공
- 빠른 원자 모델 테스트 시간
- 테스트 스크립트의 재사용성 향상

본 논문은 2장에서 DEVS 형식론 및 이산 사건 시스템을 소개하고, 3장에서 스크립트 기반 테스트 환경을 제공하는 제안 시스템을 보여준다. 그리고 4장에서 실험 결과를 분석하고, 5장에서 결론 및 향후 계획에 관해 설명한다.

II. Background and Related Work

본 장에서 2.1장은 배경인 DEVS 형식론에 대해서 소개하고, 2.2장은 DEVS 형식론을 사용한 기존 시스템들을 소개한다.

1. DEVS Formalism

DEVS 형식론은 일반적인 시스템들을 분석하기 위해 계층적이고 모듈화 개별 이벤트 모델을 표현하는 이론적이고 잘 정의된 수단이다. DEVS는 모델의 재사용성(model reusability)과 운용 의미론(operational semantics)을 포함하고 있어 실제 시스템의 실행을 묘사할 수 있다. 시스템의 묘사(행위)와 구조는 원자와 결합 모델들로 표현한다.

DEVS에서 원자 모델은 한 시스템의 입력과 출력, 상태들, 시간, 그리고 함수들을 갖는다. 시스템의 함수들은 현재 상태와 입력에 따라 다음 상태와 출력을 결정한다. 원자 모델의 형식은 다음과 같다.

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, t_a \rangle$$

위 수식에서 X는 이산사건 입력 집합, S는 일련의 이산사건 상태의 집합, Y는 이산사건 출력 집합이다. δ_{int}는 내부 상태 천이 함수, δ_{ext}는 외부 상태 천이 함수, λ: 외부 상태 천이 함수, t_a는 시간 진행 함수이다.

결합 모델의 구조는 다음과 같다.

$$CM = \langle X, Y, \{M_i\}, EIC, EOC, IC, select \rangle$$

위 수식에서 X는 이산사건 입력 집합, Y는 이산사건 출력 집합, $\{M_i\}$ 모든 이산사건 컴포넌트 모델들의 집합이다. *EIC*는 외부 입력 연결 관계, *EOC*는 외부 출력 연결 관계, *IC*는 내부 연결 관계, *select*는 같은 시간에 존재하

는 사건을 발생하는 모델들에 대한 선택 함수이다.

이러한 결합 모델은 더 큰 결합된 모델과 합쳐질 수 있거나 더 복잡한 모델을 만들기 위해 몇몇 원자 모델들과 결합할 수 있다[13]. 그리하여 이 형식론을 사용하여 다양한 분야에 적용되고 있으며 다양한 프로그램 언어로 DEVS 시스템들이 개발되었다.

2. Existing Discrete Event Simulation System

DEVS의 이산 사건 시스템들은 행위와 계층 구조를 갖는 원자와 결합 모델들을 개발하기 위해 객체지향 프로그래밍을 지원한다. DEVS 시스템들은 DEVS 모델들을 설계 및 구현하기 위해 객체지향 모델링 방법론 기반의 상속 메커니즘과 재사용성을 갖는다.

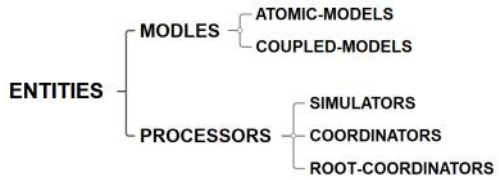

Fig. 1. Basic Class Hierarchy of DEVS System

Fig. 1은 DEVS 시스템의 기본 클래스 계층 구조를 보여준다. 이 시스템의 최상위 계층은 ENTITIES이고, MODELS와 PROCESSORS의 두 서브 클래스를 갖는다. MODELS는 모델링을 위해 사용하는데, ATOMIC-MODELS와 COUPLED-MODELS가 있다. 개발자는 ATOMIC-MODELS를 사용할 경우 시스템의 행위들을 원자 모델로 표현할 수 있고, COUPLED-MODELS를 사용할 경우 결합 모델을 정의할 수 있다. PROCESSORS는 시뮬레이션을 위해 사용하는데, SIMULATORS, COORDINATORS, ROOT-COORDINATORS가 있다. 이 클래스의 기본 계층 구조를 유일하게 사용하는 대표적인 이산 사건 시스템은 DEVS-Scheme, DEVAJava, DEVS-Python 등이 있다.

DEVS-Scheme은 DEVS 형식론이 개발된 이후 가장 처음에 개발된 시스템이다. 이 시스템은 Lisp 기반 객체 지향 프레임워크로써, 계층적이고 모듈러한 이산 사건 모델 및 시뮬레이션 환경을 제공한다[1, 3, 6]. DEVS-Scheme 시스템은 애리조나 대학의 인공지능 및 시뮬레이션 연구소에서 개발되어 객체 지향 상위 집합인 SCOOPS (Scheme Object-Oriented Programming System)를 기반으로 구축되어 꾸준히 사용되었다[1, 6, 7].

DEVSJava는 Java 프로그래밍 언어를 사용하여 복잡한 시스템의 이산 사건 모델링 및 시뮬레이션을 위한 시스템이다. DEVSJava는 목표 시스템을 분석하기 위해 구성 요소 동작뿐만 아니라 시스템 구조에 대한 정보도 함께 통합할 수 있다. DEVSJava에서 원자 및 결합 모델은 독립 실행형 애플리케이션 및 애플릿으로 모델링 및 시뮬레이션할 수 있다. 이 시스템은 웹 브라우저를 사용하여 어디에서나 시뮬레이션 모델을 실행할 수 있을 뿐만 시뮬레이션도 가능하다.

최근에는 Python 언어로 DEVS 시스템을 개발했는데[10, 12, 14], 그중 DEVS-Python은 Python 기반으로 모델링하고 시뮬레이션한다. Python을 사용하는 이 시스템에서는 시스템의 구성을 기본 클래스 계층 구조에 따라 정의한다. 또한, Python의 장점을 그대로 이어받아 문법들이 단순하여 초보자도 쉽게 모델링이 가능하며 풍부한 라이브러리를 사용할 수 있다[10]. 상기 이산 사건 시스템들 외에도 ADEVS, DEVSim++ 등이 있다.

III. DEVS Model Test

이산 사건 시스템 중 하나인 DEVS는 다양한 언어로 모델들을 개발하고 있다. DEVS 원자 모델은 4개의 함수와 3개의 집합으로 구성된다. 이 원자 모델은 시뮬레이션 모델의 최소 단위로써 시뮬레이션의 행위를 표현한다. 원자 모델의 행위는 외부상태천이함수, 내부상태천이함수, 출력함수로 구성되고, 각 함수 실행 후 원자 모델의 상태 변수는 변경된다. 원자 모델은 함수들의 실행에 따라 일반 원자 모델(입력과 출력 모두 수행), 데이터 발생을 위한 원자 모델(출력만 수행), 데이터 수집 및 분석을 위한 원자 모델(입력만 수행)로 분류할 수 있다. 3가지 원자 모델의 유형과 모델의 함수 실행 관계는 Table 1과 같다.

Table 1. DEVS models' Functions Relationship

Type of Atomic Model	External Transition Function	Internal Transition Function	Output Function
General Model	○	○	○
Data Generation Model	-	○	○
Data Collection and Analysis Model	○	-	-

○ : Execution
- : Function can be executed, but state variables are not changed

Fig. 2는 DEVS 원자 모델의 가장 기본인 P(processor) 모델의 타이밍 다이어그램을 보여준다. 이 타이밍 다이어

그램의 X, Y는 입력과 출력을 나타내고, S는 모델의 상태들(phase, sigma)을 보여준다.

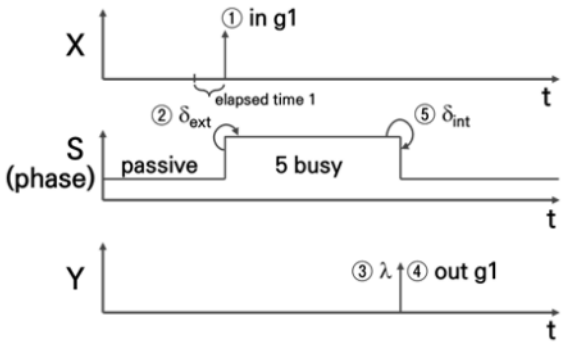

Fig. 2. Timing Diagram of P Atomic Model

원자 모델 테스트를 위해 외부상태천이함수(δ_{ext}) 명령어는 다음과 같다.

- 명령어: send <atomic-model> inject <port> <value> <elapsed-time>
- 사용 예: send p inject in g1 1
- 설명: 경과 시간 1에 in 포트에 외부 이벤트 "g1" 도착(Fig. 4. ①)
- 출력(Fig. 4 ②): (5 busy x1 5)

출력함수(λ)의 명령어는 다음과 같다.

- 명령어: send <atomic-model> output?
- 사용 예: send p output?
- 설명: 출력함수 실행(Fig. 4. ③)
- 출력(Fig. 4. ④): (out x1)

내부상태천이함수(δ_{int})의 명령어는 다음과 같다.

- 명령어: send <atomic-model> int-transition
- 사용 예: send p int-transition
- 설명: 내부상태천이함수 실행(Fig. 4. ⑤)
- 출력: (INF passive x1 5)

Fig. 3은 P 모델의 외부상태천이함수를 수동 테스트한 결과를 보여준다.

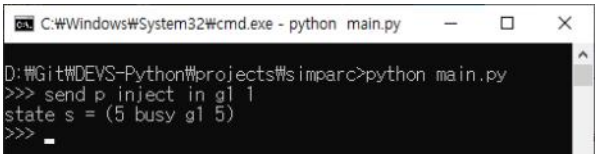

Fig. 3. Example of Manual Test Result of P Atomic Model

2.2장에서 제시한 기존 DEVS 시스템 중에서 DEVS-Scheme과 DEVS-Python은 원자 모델 검증을 수동 테스트로 진행한다. 이는 모델 검증을 위해 오랜 시간이 걸릴 뿐만 아니라 테스트 코드를 재사용하기 어렵다. 최근에는 이들의 문제를 해결하기 위해 테스트 주도 개발(Test Driven Development; TDD) 기반의 애자일 기법[15, 16]이 사용되고 있다.

IV. Proposed System

본 장은 스크립트 기반 원자 모델 테스트를 제공하는 이산 사건 시스템을 제안한다. 4.1장은 본 시스템의 개요를 소개하고, 4.2장은 제안 시스템의 절차를 보여준다. 4.3장과 4.4장은 원자 모델을 위한 수동 테스트와 스크립트 기반 테스트에 관해 설명한다.

1. Overview

우리의 제안 시스템은 모델링 시간 단축을 위해 DEVS-Python에서 스크립트 기반 원자 모델 테스트를 제공한다. Fig. 4는 제안 시스템의 개요를 보여준다. 개발자는 목표 원자 모델을 구현한 후 그 모델의 검증하고 확인할 스크립트를 작성한다. 이후 모델 베이스(model base)에서 한 원자 모델을 선택한 후 작성된 스크립트를 실행한다. 그 모델의 검증과 확인 결과는 보고서로 출력되어 개발자가 확인할 수 있다.

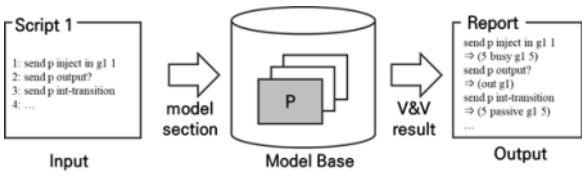

Fig. 4. Proposed System Overview

2. Execution Procedure

제안 시스템은 구현된 원자 모델들의 짧은 검증 시간을 보장하기 위해 스크립트 기반 자동 테스팅 도구를 제공한다. 우리 시스템에서 개발자는 원자 모델을 설계 및 구현하고 테스트 스크립트를 실행한 후 이를 검증한다.

Fig 5는 제안 시스템에서 원자 모델의 수행 절차를 보여준다. 모델링의 절차는 개발자와 DEVS 엔진(engine) 레벨로 나뉜다. 모델 개발자는 초기에 원자 모델을 설계(①) 및 구현(②)한다. 이후 구현된 모델에 따라 테스트 스크립트를 작성한다(③). 개발자는 원자 모델 테스트를 실

행한 후 수동과 자동 테스트 모드를 선택한다(④). 이때 자동 테스트를 선택할 경우 작성된 스크립트 파일을 연다(⑤). DEVS 엔진은 테스트 스크립트에 따라 모델을 검증(⑥)한 후 DEVS 모델의 여부를 확인한다(⑧). 만약 ④단계에서 수동 테스트 모드로 실행될 경우 개발자는 테스트 명령어를 직접 입력(⑦)한 후 그 모델을 검증할 수 있다.

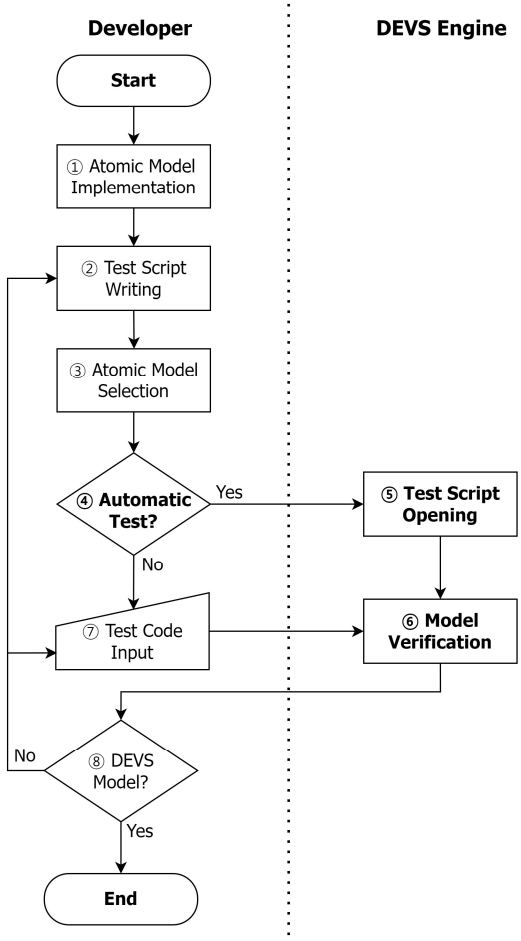

Fig. 5. Atomic Model Modeling Procedure

3. Script-based Test of Atomic Models

제안 시스템에서는 원자 모델의 수동 테스트뿐만 아니라 스크립트 기반 테스트도 제공한다. 개발자는 한 모델을 정의한 후 JSON 형식으로 테스트 스크립트를 작성할 수 있다.

Table 2. JSON-based Test Script Grammar

```
1: {
2:     "model name": {
3:         "1": "test command1",
4:         "2": "test command2",
5:         "3": "test command3",
6:         …
7:     },
8:     …
9: }
```

Table 2는 원자 모델의 JSON 기반 스크립트 템플릿을 보여준다. 개발자는 스크립트를 작성할 때 테스트할 모델의 이름을 기재(2줄)하고 원자 모델에 보낼 스크립트를 순서대로 작성한다(3~6줄). 테스트 명령어는 스크립트 순차 번호와 명령어가 한 쌍을 이룬다. 이 테스트 명령어는 개발자를 통해 거의 무한대로 작성될 수 있다. Table 3은 P 모델의 테스트 스크립트 한 예를 보여준다.

Table 3. Example of Test Script of P Atomic Model

```
{
    "P": {
        "1": "send p inject in g1 1",
        "2": "send p output?",
        "3": "send p int-transition"
    }
}
```

Table 3의 테스트 스크립트는 Fig. 6과 같이 실행된다. Fig. 6은 별도의 사용자 입력 없이 main.py를 통해 P 모델과 JSON 파일을 연 후 자동으로 실행되는 모습이다.

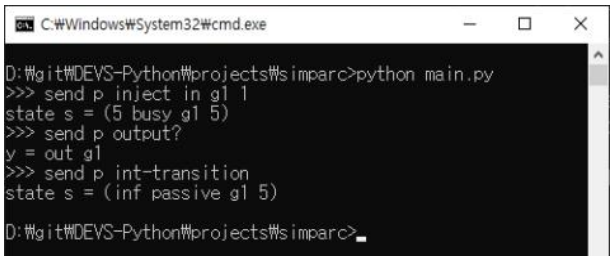

Fig. 6. Example of Execution Result

V. Experimental Result

본 장에서는 제안 시스템의 효율성을 평가하기 위해 아래 두 기법을 사용하여 성능을 평가한다.

- 수동 테스트 기법
- TDD 기법(Unitest)

실험 환경은 Windows 10에서 CPU가 Intel(R) Core i5-4570 3.20GHz, RAM 8GB 등이다. 본 실험을 위해 Visual Studio Code (VSCode)를 사용하였다. 수동 테스트를 위해서는 이산 사건 시뮬레이션 시스템(DEVS-ObjC 등)에서 제공하는 수동 테스트와 제안 시스템의 스크립트 기반 자동 테스트의 실행 시간을 비교한다. 수동 테스트는 60 WPM (Word Per Minute)[17] 기준으로 진행한다. 다만 수동 테스트는 명령어를 반복하여 타이핑할수록 WPM이 올라갈 수도 있지만, 이는 고려하지는 않았다. P 모델을

대상으로 외부상태천이함수, 출력함수, 내부상태천이함수의 한 사이클 글자 수는 총 60글자(빈칸 포함)이다. 따라서 P 모델을 수동 테스트로 1사이클을 진행할 경우 60 WPM, 곧 1분이 소요된다. 1사이클(cycle)은 P 모델의 외부상태천이함수, 출력함수, 내부상태천이함수를 각각 실행한다.

Table 4. Experimental Results of Manual Test Versus Script-based Test

Test Type / Time	Manual Test	Script-based Test	
	1 Cycle	1 Cycle	10 Cycle
Execution Time	60 sec	4 ms	24 ms

Table 4는 수동 테스트와 제안 시스템의 스크립트 기반 자동 테스트(1회, 10회 연속)의 실험 결과를 보여준다. 기존 시뮬레이터는 1사이클의 수동 테스트 소요 시간은 60초가 걸린다. 반면, 제안 시스템의 소요 시간(10회 평균)은 1 사이클 실행 시 약 0.004초, 10사이클 실행 시 약 0.0238초가 소요된다. 그리하여 제안 시뮬레이터는 수동 테스트 소요 시간인 1분 대비 0.004초의 소요로 실행 시간이 상당히 향상되었음을 확인하였다.

제안 시스템이 TDD 기법과의 비교를 위해서는 DEVS-Python을 사용하여 테스트 실행 시간을 확인한다. TDD 기법은 파이썬 유닛 테스트(unitest)[18]를 사용한다. 본 실험은 시범적으로 사용하는 P 모델에 대해 우리의 스크립트 기반 자동 테스트와 파이썬 유닛 테스트를 각각 실행하고 각 실행 시간을 비교한다.

Fig. 7은 두 테스트 도구를 사용하여 사이클 대비 실행 시간을 보여준다. Fig. 7에서 보듯이 P 모델을 5번 테스트할 때 유닛 테스트는 11ms를 실행한 반면, 우리의 스크립트 기반 테스트는 8ms에 실행했다. 10번 테스트할 때 역시 유닛 테스트는 27ms 동안 실행했지만, 제안 시스템의 테스트는 24ms 동안 실행했다. 본 실험 결과, 제안 시스템이 기존 테스트 기법에 비해 최소 20% 이상 시간을 줄여 효율성이 뛰어나다는 것을 볼 수 있다.

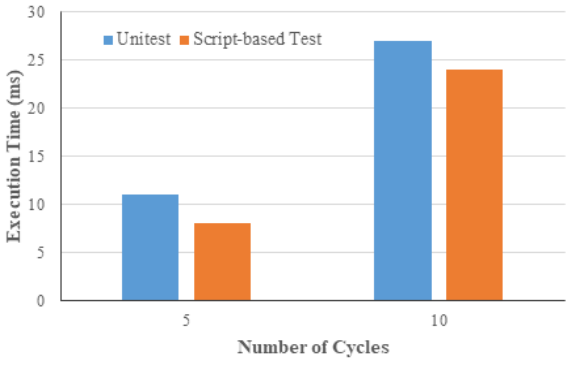

Fig. 7. Execution time versus number of cycles

VI. Conclusions

본 제안 시스템은 DEVS-Python에서 원자 모델의 검증 시간을 줄이기 위해 스크립트 기반 테스트 환경을 제공한다. 기존 시스템은 원자 모델 검증을 위해 개발자가 콘솔에 테스트 명령어를 직접 입력(수동 테스트)해야 했지만, 제안 시스템은 사전에 작성된 스크립트를 활용하여 모델의 동작을 빠르게 확인할 수 있다. 그리하여 우리의 시스템은 10사이클 실행 시 24ms로 빠르게 실행하였으며, TDD 기반 유닛테스트와 비교했을 때도 20% 정도 실행 속도가 빨랐다. 향후 연구는 스크립트 기반 테스트베이스(testbase)를 만들어 여러 원자 모델을 효과적으로 테스트할 수 있는 환경을 개발할 것이다.

ACKNOWLEDGEMENT

This work was supported by the Technology development Program (S2837815) funded by the Ministry of SMEs and Startups (MSS, Korea)

REFERENCES

[1] T. G. Kim, "Hierarchical development of model classes in the DEVS-scheme simulation environment," Expert Systems with Applications, vol. 3, no. 3, pp. 343-351, 1991.

[2] S. M. Nam and T. H. Cho, "Context-Aware Architecture for Probabilistic Voting-based Filtering Scheme in Sensor Networks," IEEE Transactions on Mobile Computing, vol. 16, no. 10, pp. 2751-2763, 2017.

[3] B. P. Zeigler, "Hierarchical, modular discrete-event modelling in an object-oriented environment," Simulation, vol. 49, no. 5, pp. 219-230, 1987.

[4] G. A. Wainer, K. Al-Zoubi, D. R. Hill, S. Mittal, J. L. R. Martín, H. Sarjoughian, L. Touraille, M. K. Traoré, and B. P. Zeigler, “An introduction to devs standardization,” In Discrete-Event Modeling and Simulation, CRC Press, pp. 393-425, 2018.

[5] Y. Van Tendeloo, H. Vangheluwe, “An evaluation of DEVS simulation tools,” Simulation, vol. 93, pp. 103-121, Feb. 2017.

[6] B. P. Zeigler, "DEVS-SCHEME: a LISP-based environment for hierarchical, modular discrete event models," Dep. Elec. Comput. Eng., Univ. Arizona, Tucson, AZ, Tech. Rep, 1986.

[7] M. W. Aiken and G. J. A. S. S. D. Hayes, "A DEVS-Scheme simulation of an electronic meeting system," vol. 20, no. 2, pp. 31-39, 1989.

[8] H. S. Sarjoughian and B. Zeigler, "DEVSJAVA: Basis for a DEVS-based collaborative M&S environment," Simulation Series, vol. 30, pp. 29-36, 1998.
[9] CosMoSim, 2019, https://sourceforge.net/projects/cosmosim/
[10] S. M.. Nam. DEVS-Python, 2022, https://github.com/sumannam/DEVS-Python.
[11] J. Nutaro, ADEVS, 2022. https://web.ornl.gov/~nutarojj/adevs/
[12] K. Tag Gon et al., "DEVSim++ Toolset for Defense Modeling and Simulation and Interoperation," The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, vol. 8, no. 3, pp. 129-142, Jul. 2010.
[13] B. P. Zeigler, Object-oriented simulation with hierarchical, modular models: intelligent agents and endomorphic systems. Academic press, 2014.
[14] Capocchi, L., Santucci, J. F., Poggi, B., Nicolai, C, “DEVSimPy: A collaborative python software for modeling and simulation of DEVS systems”. In 2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 170-175, June 2011.
[15] V. Thimothe, L. Capocchi, and J. F. Santucci, "DEVS models design and test using AGILE-based methods with DEVSimPy," in 26th European Modeling and Simulation Symposium (EMSS), pp. 563-569, 2014.
[16] K. J. Hong and T. G. Kim, "DEVSpecL: DEVS specification language for modeling, simulation and analysis of discrete event systems," Information and Software Technology, vol. 48, no. 4, 2006, pp. 221-234.
[17] R. W. Soukoreff and I. S. MacKenzie, "Metrics for text entry research: An evaluation of MSD and KSPC, and a new unified error metric," in Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 113-120, 2003.
[18] Python Unit testing framework, 2020, https://docs.python.org/3/library/unittest.html.

Authors

Su-Man Nam received the M.S. and Ph.D. degree in Electrical and Computer Engineering from SungkyunkwanUniversity, Korea, in 2013 and 2017 respectively. Dr. Nam joined a researcher in the Department of Biomedical Informatics, Ajou University, Suwon, Korea, in 2018. He is currently a head of research institute in DuDu Information Technologies, Inc., Seoul, Korea. He is interested in modeling and simulation, digital twin, WSN, and IoT.