한국군사과학기술학회지 제12권 제5호, pp. 590~600, 2009년 10월

학술논문 정보기술 부문

스크립트 임베딩을 활용한 수중운동체 M&S 전술처리기의 기능 확장

The Functional Extension of the Underwater Vehicle Modeling and Simulation Tactics Manager using the Script Embedding Method

손 명 조* 김 태 완** 나 영 인***
Myeong-Jo Son Tae-Wan Kim Young-In Nah

Abstract

In the simulation of underwater vehicles such as a submarine or a torpedo, various type of simulations like an engineering level simulation for predicting the performance precisely and an engagement level simulation for examining the effectiveness of a certain tactic is required. For this reason, a tactics manager which can change the behavior of a simulation model according to external tactics is needed. In this study the tactics manager supporting a script language and engine which can represent various tactics and can help users define external input tactics for the tactic manager easily is suggested. Python and Lua which are representative among script languages have been compared and analyzed from the viewpoint of a tactic manage, and the tactic manger using the script engines of those script languages was implemented. To demonstrate the effectiveness of the tactic manager, a target motion analysis simulation of the warfare between a submarine and a surface ship.

Keywords : Modeling and Simulation(모델링과 시뮬레이션), Tactics Manager(전술처리기), Script Embedding(스크립트 임베딩), Target Motion Analysis(표적기동분석), Underwater Vehicle(수중운동체), Lua(루아), Python(파이썬)

1. 서 론

잠수함, 어뢰, 무인잠수정과 같은 수중운동체(Underwater Vehicle)는 개발에 막대한 예산과 기간이 소요되기 때문에 설계와 건조 시 겪을 수 있는 시행착오로 인한 개발위험성을 최소화하고 최적설계를 수행하기 위하여 모델링 & 시뮬레이션을 통한 성능예측이 필수적이다. 또한, 이러한 수중운동체의 운용에 있어서 다양한 전술에 따른 효과를 예측하기 위해서도 모델링 & 시뮬레이션 기술이 필요하다.

수중운동체의 다양한 체계/부체계의 성능을 분석하기 위해서는 대상의 수학적 또는 공학적인 모델링을 통하여, 이산시간 기반으로 시뮬레이션을 진행하는 공학급 시뮬레이션이 활용된다. 이러한 공학급 모델의 간략화 및 추상화를 통해 생성한 교전급 모델을 활용

† 2009년 4월 16일 접수~2009년 7월 31일 게재승인
* 서울대학교 조선해양공학과 대학원(Seoul National University)
** 서울대학교 조선해양공학과 및 해양시스템공학연구소(Seoul National University and RIMSE)
*** 국방과학연구소 제6기술연구본부(ADD)
책임저자 : 김태완(taewan@snu.ac.kr)

하여, 다양한 전술을 시험분석하고, 새로운 전술을 개발할 수 있는 교전급 시뮬레이션은 이산사건 기반으로 시뮬레이션이 진행되며, 동일 모델을 이용하여 전술만을 다양하게 변화시켜가며 반복적인 시뮬레이션을 진행하게 된다. 이러한 교전급 시뮬레이션을 위해 전술처리기(Tactics Manager)가 제안되었다[2,4].

현실세계에서는 함장의 결정에 의해 잠수함이 운용되지만 시뮬레이션에서는 사용자가 사전에 입력해놓은 명령 또는 전술 데이터에 따라서 모델이 동작하게 된다. 전술처리기는 미리 입력한 전술에 따라서 모델이 동작방식을 동적으로 결정할 수 있도록 도와주는 역할을 담당한다. 전술처리기가 구현되어 있지 않은 시뮬레이션 모델의 경우, 동작순서나 전술은 모델 내부에 고정적으로 정의되어 전술이 변경될 때마다 모델이 수정된 후, 재컴파일 되어야 한다. 이를 개선하기 위해, 전술처리기는 전술을 외부에서 파일로 입력받고, 입력된 전술을 분석하여 모델의 동작을 제어한다. 즉, 전술이 변경되어도 모델을 수정하지 않고 전술 정의 파일만 수정하여 다양한 시나리오를 시뮬레이션 할 수 있는 장점을 가지고 있다.

본 연구에서는 전술처리기에 사용되는 전술정의 파일을 사용자가 쉽게 정의할 수 있고, 다양하게 전술을 표현할 수 있는 스크립트 언어를 사용한 전술정의방법을 연구하고, 스크립트 엔진을 활용한 전술처리기를 구현하였다. 시뮬레이션을 위한 모델은 이산사건 시뮬레이션에 널리 사용되는 DEVS 형식론(Discrete EVent System specification formalism)[3,15,16]을 이용하여 구현하였다. 구현된 모델과 연동하는 전술처리기의 효과를 검증하기 위해서 잠수함과 수상함 교전 시뮬레이션 중, 잠수함의 표적기동분석(Target Motion Analysis)에 적용하였다.

본 논문의 구성은 다음과 같다. 2장에서는 스크립트 기반 전술처리기의 개념과 동작원리에 대하여 설명하고, 3장에서는 전술처리기의 개념을 적용한 표적기동분석 시뮬레이션과 실행결과를 기술한다. 마지막으로 4장에서는 결론과 향후 연구방향을 제시한다.

2. 전술처리기

가. 전술처리기의 개념

전술처리기는 “If(상태조건) Then(전술 1), Else(전술 2)”와 같이, 의사결정이 필요한 경우, 미리 정의된 규칙에 따라 전술을 결정한다. 즉, 모델이 특정 상태조건을 만족하면 전술 1을 수행하고, 그렇지 않다면 전술 2를 수행하도록, 의사를 결정하는 것이 전술처리기의 기본 개념이다. 이러한 전술처리기는 같은 모델을 이용하여, 전술을 결정하는 방법만을 다양하게 변화하면서 반복적으로 시뮬레이션을 시행하고 특정 전술의 효과도를 분석하는데 특화된, 모델 구조와 연계된 전술처리 엔진이다.

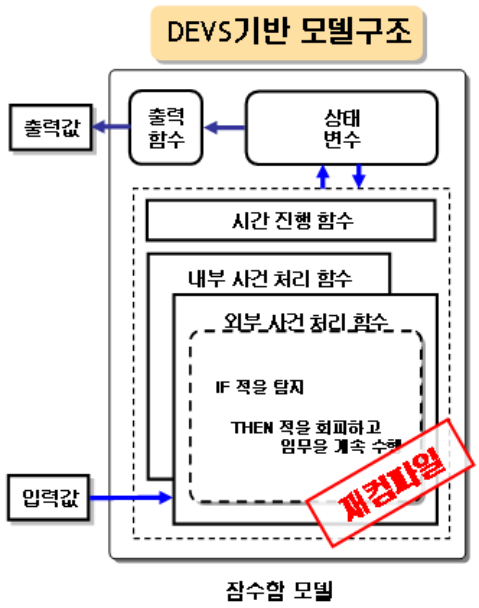


Fig. 1. 전술처리기가 없는 기존모델구조

Fig. 1과 같이, 이산사건을 처리하기 위해, 체계적으로 7가지 구성요소(입력, 출력, 내부 사건 처리, 외부 사건 처리, 시간진행, 상태변수, 출력함수)를 가지는 DEVS 형식론을 따르는 모델구조에서, 의사결정이 필요한 모델의 동작원리는 외부사건 처리 함수 내부에 하드코딩된다. 이 경우, 의사결정 방식이나 상태조건이 조금만 변경되어도 모델 뿐만 아니라, 전체 시뮬레이션이 재컴파일되어야 한다.

전술처리기를 활용하면, Fig. 2와 같이 모델 외부에, 전술을 테이블 형식 또는 스크립트 형식으로 파일로 정의하고, 의사결정이 필요할 때마다 전술처리기가 이를 호출하여 처리한다. 의사결정 부분이 변경되더라도, 모델과 전체 시뮬레이션은 재컴파일될 필요가 없고, 실행만 하면 변경된 의사결정 방법이 적용된 결과를 얻게 된다.

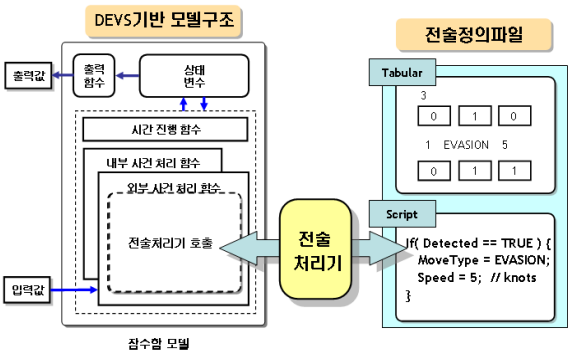


Fig. 2. 전술처리기를 사용한 모델구조

나. 전술 정의

본 연구에서 제시하는 전술처리기는 다양한 전술의 표현과 명확한 가독성을 위해, 트리구조와 같은 계층적인 전술정의를 지원한다. 이를 활용하여, 3장의 잠수함의 표적기동분석 시뮬레이션을 위한 전술을 기본단계, 탐지 및 접근 단계, 공격 단계의 세 단계로 분류하였고, 각 단계별로 구체적인 세부 전술을 정의하였다. 각 단계로 분기하기 위해서는 모델로부터 전달받은 상태변수와 전술정의파일에 정의된 상태조건를 비교하여, 상태변수가 상태조건을 만족하여야한다. 예를 들어, 본 시뮬레이션에서는 각 단계로 분기하기 위한 상태변수로는 탐지여부, 적과의 거리를 사용한다. 따라서, 모델은 항상 탐지여부와, 적과의 거리 속성값을 전술처리기로 전달해야한다. 전술의 상태조건과 전술처리기의 상태변수는 그 자료구조와 순서가 일치하여야 하며, 이는 모델을 개발한 후, 전술의 정의할 때 미리 약속되어 있어야 한다. 전술정의를 할 때, 새로운 상태조건이 추가되어, 모델로부터 추가적인 상태변수가 요청되어야 할 경우는 모델의 추가적인 수정이 필요하다.

```
If (탐지 = = TRUE)
Then { IF (적과의 거리 ! = NULL)
          Then (공격 단계)
          Else (탐지 및 접근 단계) }
Else (기본 단계)
```

'기본 단계'는 정해진 구역을 미리 정의된 일정한 패턴으로 기동하며 초계하는 방책임무(Barrier Mission)를 수행하는 것이다. 이때 적의 존재가 탐지되고 적과의 거리를 모르는 상태이면 표적기동분석을 수행하는 '탐지 및 접근 단계'로 분기한다. 적을 처음 탐지하게 되면, 적과의 거리는 알 수 없기 때문에, 무조건 '탐지 및 접근 단계'로 분기한다. 표적기동분석을 수행하여 적과의 거리를 알고 있는 상태라면 '공격 단계'로 분기한다.

각 단계는 다시 구체적인 전술로 정의된다. 그 중 한 예로, '탐지 및 접근 단계'는 아래와 같다.

```
If (LEG 번호 = = 1)
Then (100% 확률  LEG Type = "POINT")
Else If (LEG 번호 = = 2)
Then (70% 확률 LEG Type = "LEAD"
        30% 확률 LEG Type = "LAG")
Else {If(LEG Type = = "LEAD")
          Then(50% 확률 LEG Type = "POINT"
                  50% 확률 LEG Type = "LAG")
   Else(50% 확률 LEG Type = "POINT"
        50% 확률 LEG Type = "LEAD")}
```

즉, 탐지 및 접근단계 전술은 상태변수로 현재 LEG 번호(TMA 과정 중 몇 번째 단계의 기동인지를 뜻함), 선행 LEG Type(TMA 과정 중, 본 기동 이전의 기동 종류)를 사용하고, 전술은 확률에 의해 분기된다. 본 시뮬레이션에는 수중환경 등의 예측불가능한 외부환경요소를 고려하고 있지 않고 있어 결정론적인 방식(Deterministic)을 이용하면, 시뮬레이션을 반복수행하여도, 같은 결과가 도출되기 때문에, 확률론적인 방식(Stochastic)으로 결과의 다양성을 보장하였다. 즉, 모델로부터 같은 상태변수값이 입력되어, 동일한 전술로 분기할 수 밖에 없는 상황에서라도, 최종적으로는 전술처리기에서 난수(Random Number)를 생성하여, 확률에 따라 다양한 전술을 결정할 수 있도록 하였다. 이러한 확률값은 전술정의 전문가의 지식에 경험하여, 정확한 수치가 입력되면 더욱 정교한 시뮬레이션 결과를 보장하며, Fuzzy와 같은 인공지능 기법을 사용한다면, 더욱 현실에 가까운 전술결정이 가능하다. 시뮬레이션에 사용된 전체 전술 정의와 그에 대한 설명은 선행 연구를 따른다[4].

다. 전술 정의 파일

이러한 전술을 실제 전술처리기를 사용하여 모델과 연동시키려면 전술을 파일로 정의하는 것이 필요하다. 기존 연구[2]에서는 DEVS 또는 이산시간 기반의 모델

구조가 없는 시뮬레이션 모델 내부에 전술처리기 함수를 정의하고, 외부에 고정된 테이블 형식의 텍스트 파일에 기정의된 값을 기입하는 것으로써 전술을 정의하였다. 모델과 분리된 전술처리기 개념이 제안된 선행 연구[4]에서는, DEVS기반의 시뮬레이션 모델구조와 연계하여, 전술이 결정될 필요가 있을 경우, DEVS의 외부사건함수에서 외부 모듈로 구현된 전술처리기를 호출하고, 전술처리기에서는 테이블 형식의 전술정의파일을 읽어들여, 전술을 결정하고 모델로 이를 전달하였다. 이때, 전술정의 파일은 기존연구[2]와는 달리, 상태조건의 개수와 자료형(Data Type) 등을 결정할 수 있는 유연하고 가변적인 테이블 형식을 전술정의파일로 채택하였다. Fig. 3은 '2-나'절에 정의된 단계별 전술을 이러한 가변적인 테이블 형식으로 정의한 파일이다.

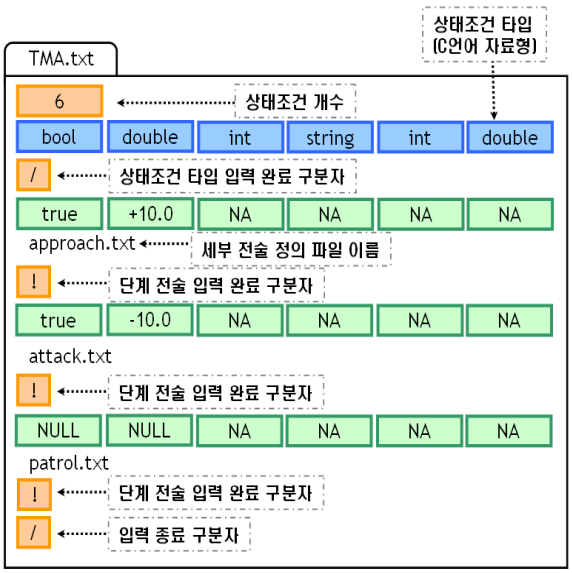


Fig. 3. 전술 정의 파일 구조

상태변수의 개수와 자료형은 변경가능 하더라도, 그에 해당하는 상태조건 값은 bool 타입의 경우, true/false, 정수형(Int)은 기정의된 어뢰의 개수 또는 표적 기동분석 기동 순서 등과 같은 상태조건값을 고정적으로 기입을 하는 형태로 전술작성이 완성된다. 이와 같이, 테이블 형식의 전술정의 파일은 고정적인 형태든, 가변적인 형태든, 해당되는 빈 칸에 사전에 약속된 데이터 값의 변경만 가능하다는 점에서 동일하게, 유연하지 못하다. 이러한 형태로는 구체화된 전술을 표현하기 어려울뿐더러, '2-나'절에 표현된 전술을 다시 테이블의 형태의 전술로 가공하여야 한다. 이러한 가공된 테이블 전술정의파일을 '2-나'절에 표현된 전술 형태로 해석하는 것은 용이하지 않다.

'2-나'절에서는 '3'장의 case study 시뮬레이션 시나리오로부터 전술 요소를 추상화하여, 사람의 언어와 가까운 간단한 문법의 프로그래밍 언어로 표현하였다. 이러한 개별 전술 요소를 다시 테이블 형태로 가공하는 테이블 전술정의파일의 단점을 개선할 수 있는 방법이 스크립트 기반 전술정의법이다. 즉, 범용 스크립트 언어를 사용하여 전술을 표현할 경우, pseudo code와 같은 전술도 큰 수정 없이, 각 스크립트 언어 문법 형식만을 지키는 선의 간단한 수정만으로 전술정의가 가능하다.

Fig. 4, Fig. 5는 Fig. 3에서 나타낸 테이블 형식의 전술정의 형식과 동일한 전술 정의 정보를 각기 다른 스크립트 언어를 사용하여 표현하고 있다. 테이블 형식에서는 상태조건의 개수를 나타내는 '6'의 의미와 상태조건의 자료형을 뜻하는 'bool'과 'double' 등이 의미하는 것이 무엇인지 파일에는 나타낼 수 없지만, 스크립트 형식에서는 num_of_decision_cond(상태변수의 개수), types_of_decision_cond(상태변수의 자료형) 등으로 명확하게 표현할 수 있다. 또한, 첫 번째 상태변수는 'bool' 자료형으로, 'detect'(적함 탐지 여부)이며, 두 번째 상태변수는 'double' 자료형으로, 'range'(적과의 거리)임을 가시적으로 표현이 가능하다. 이를 통해, 테이블 형식에서 상태변수의 의미가 기정의(Predefine) 또는 약속에 의해 손실되는 단점을 극복할 수 있다. 이러한 점이 스크립트 전술정의 파일의 가독성을 보장하여, 전술정의 파일을 읽는 것만으로도 그 전술을 직관적으로 명확하게 파악할 수 있다.

```
num_of_decision_cond = 6
types_of_decision_cond={"bool", "double", "int", "string", "int", "double"}
name_of_func="TMA"

function TMA(detect, range, CrrLeg, PrvLeg , num_of_torp, batt )
if detect==true then
        if range ~= nil then
        return "attack.lua"
        else
        return "approach.lua"
        end
else
return "patrol.lua"
end
end
```

Fig. 4. 스크립트 기반 전술 정의 파일(Lua)

```
num_of_decision_cond = 6
types_of_decision_cond=['bool', 'double', 'int', 'string', 'int', 'double']
name_of_func="TMA"

def TMA(detect, range, CrrLeg, PrvLeg , num_of_torp, batt ):
    if detect==True:
        if range is not None:
            return "attack"
        else:
            return "approach"
    else:
        return "patrol"
```

Fig. 5. 스크립트 기반 전술 정의 파일(Python)

```
num_of_decision_cond = 6
types_of_decision_cond=['bool,'double','int',
                        'string', 'int', 'double']
name_of_func="Extended_TMA"

def Extended_TMA(detect, range, current_leg,
                 previous_leg, num_of_torpedo, battery):

    if detect == true:
        leg type=Determine_Moving_Action
        (detect, range, current_leg, previous_leg)
    else:
        return "No_Action"

def Determine_Moving_Action
    (detect, range, current_leg, previous_leg):
     tactic_moving=[]
     k0=(FLV_Distance_Warning(range),0,"POINT")
     k1=(FLV_Distance_Danger(range),1,"LEAD")
     k2=(FLV_Distance_NoThreat(range),2,"LAG")

    #하나의 이중배열로 전술을 저장
     tactic_moving=[k0,k1,k2]

    #배열 내에서 순차적 정렬
     tactic_moving.sort(reverse=True)

    #새로운 함수에 의해 개개의 전술 평가
     for i in range(2):
         temp=temp+
         _func(tactic_moving[i][0],
     tactic_moving[i+1][0],current_leg, previous_leg)


    #적합도 평가에 의해 유일해가 구해졌을 경우
     if temp==0:
             index=0

    #적합도 평가에 의해 복수의 전술이 결정시 임의로 선택
    else:
             import random
             index=random.randrange(0,temp)

    #이중 배열 자료 구조에서 선택된 전술을 찾아내는 과정
     for j=0 in range(3):
             if tactic_moving[j][1]==index:
                     flag=j

    #선택된 전술을 반환
     return tactic_moving[flag][2]
```

Fig. 6. 스크립트 언어를 활용한 복잡한 전술 표현

Python, Lua와 같은 범용 스크립트 언어는 그 언어가 표현할 수 있는 다양한 함수정의 기능을 활용하여, 선박의 선체 구조 설계에서 여러 구조 보강재의 용이하게 생성하는 구조 모델링 정의 파일로도 활용된 연구[1]가 있는 만큼, 그 확장성을 충분히 가늠할 수 있다. Fig. 6은 이러한 가능성을 보여주기 위해 제어문, 정렬, 조건문, 계층적인 함수, 반복문 등을 이용하여 복잡한 전술을 Python으로 표현한 예이다. 이처럼 자유롭게 복잡한 전술을 표현하기 위해서는 해당 스크립트 문법에 대한 기초적인 지식의 선행습득이 요구된다.

라. 모델과 전술처리기사이의 인터페이스

전술 정보를 읽어 들여, 전술을 전술처리기에서 실제 결정하는 테이블 형식의 전술처리기와는 달리, 스크립트 기반 전술처리기는 모델로부터 속성값을 전달받고, 이를 이용해 스크립트 전술정의 파일을 스크립트 엔진을 이용하여 실행시키며[14], 스크립트 엔진에서 결정된 전술을 모델로 전달하는 역할만을 수행한다. 잠수함 모델과 스크립트 기반 전술처리기, 그리고 스크립트 형식 전술 전술정의파일 사이의 관계는 Fig. 7과 같다.

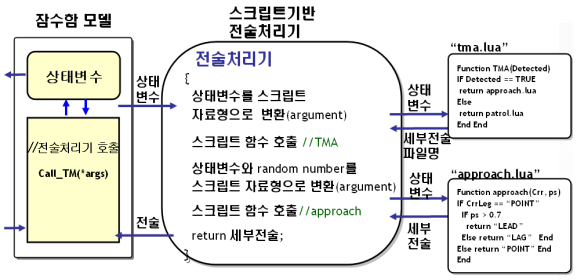


Fig. 7. 모델, 스크립트 기반 전술처리기, 스크립트 형식 전술정의파일 사이의 인터페이스 구조

Fig. 8은 스크립트 기반 전술처리기의 동작원리를 보여주고 있다. 즉, 스크립트 기반 전술처리기는 모델로부터 전술결정에 사용될 모델 속성값을 받는다. 이때, 모델에서 생성되는 속성값은 MFC에서 제공하는 CObArray 클래스 자료형으로 생성이 된다. 이는 복수의 이종 자료형으로 구성된 모델의 상태변수 값을 하나의 배열 자료형으로 사용할 수 있는 이점이 있어, 전술처리기와의 인터페이스가 간단하면서도, 정확한 기능을 수행할 수 있게 된다. 전달받은 상태변수는 Python 또는 Lua 등, 해당되는 스크립트 형식의 자료형으로 변환하여, 스크립트 전술정의 파일의 전술함수의 인자(Argument)로 사용된다. 전술처리기는 스크립트

엔진을 호출하여, 스크립트언어로 정의되어 있는 전술 정의파일을 실행한다. 스크립트 엔진을 통해, 인자로 전달된 상태변수가 상태 조건 검사를 걸쳐 단계 전술이 결정되고, 그 단계 전술에 해당되는 세부 전술정의 파일명이 전술처리기로 전달된다. 전술처리기는 확률론적인 세부전술 선택에 사용될 난수를 생성하여, 상태변수와 함께 스크립트 엔진으로 전달한다. 스크립트 엔진은 호출된 세부 전술정의파일을 실행하여, 적절한 세부전술을 결정하고, 이를 전술처리기에 전달한다. 전술처리기는 전달받은 세부전술을 C언어 자료형으로 변환한 후, 모델로 전달한다. 세부 전술은 'string'자료형으로 간단한 전술을 단어 또는 그러한 단어의 조합 형태로 전달하게 되는데, 모델에는 그러한 단어가 DEVS의 상태변수로 정의되어 있어야 하며, 그러한 상태변수에 적합한 동작원리가 내부사건처리함수에 기술되어 있어야한다.

위와 같이 C언어로 작성된 전술처리기를 개발할 때, 스크립트 언어로 작성된 함수가 호출되어 사용되는 것을 스크립트 임베딩(Script Embedding)이라 한다[6]. 반면, 스크립트 언어를 중심으로 프로그램을 개발하며, 외부프로그램(SQL, C, XML 등)의 함수를 불러서 사용하는 것을 스크립트 확장(Script Extending)이라 한다[12].

Fig. 8. 스크립트 기반 전술처리기의 동작원리

전술 결정을 직접하는 테이블 기반 전술처리기에 비해, 스크립트 기반 전술처리기는 그 역할이 상당히 축소되어, 전술처리기의 개념에 부합되지 않게 보일 수도 있다. 이는 스크립트 임베딩을 이용하여, 전술처리기의 기능을 스크립트 형식의 전술정의 파일과 스크립트 엔진으로 이양시켜 그 구조가 간단하여 졌기 때문이다. 하지만, 스크립트 언어의 API를 모델로부터 분리하여, 재사용성을 높이고 전술 수정 후 모델의 재컴파일을 없애기 위해서는 이러한 전술처리기 개념이 꼭 필요하다. 또한, 전술처리기 개발자 측면에서는 제공되는 스크립트 API를 적절히 사용하면, 테이블 번역기(Parser 또는 Interpreter)를 필수적으로 손수 구현해야하는 테이블 기반 전술처리기에 비해 구현하기가 한결 용이한 장점이 있으며, 그 코드 길이도 10분의 1 이하로 줄어든다. 전술처리기 사용자 측면에서는 본인에게 익숙한 스크립트 언어를 선택하여, 큰 어려움 없이 전술을 정의할 수 있고, 스크립트 언어의 고급 기능을 활용하여, 더욱 정교한 전술을 다양하게 표현할 수 있는 장점이 있으며, 이는 테이블 기반 전술처리기에 비해 큰 강점이다.

마. 대표적인 스크립트 언어(Lua, Python)

많은 스크립트 언어가 다양한 분야에서 사용되고 있다. 그 중, 대표적인 언어인 Python과 게임 분야에서 활발하게 사용되는 Lua를 비교분석하기로 한다. Lua는 스크립트 임베딩에 가장 많이 사용되는 언어 및 이를 실행해주는 엔진으로 브라질 Pontifical Catholic University of Rio de Janeiro의 R. Ierusalimschy의 연구팀이 1993년에 개발하고, 계속 유지, 보완하고 있다[9,10]. 반면, Python은 가장 많이 사용되고, 널리 알려진 스크립트 언어 및 이를 실행해주는 엔진이다.

Python은 초기개발(Pilot Development)에 주로 사용되며, 스크립트 확장에 강하다[12]. 즉, Python을 사용하여 프로그램을 개발하면, 다른 개발언어보다 개발기간이 대폭 줄어들며, 코드도 무척 간결해진다. 한편, Lua는 Python보다 실행속도가 빠르며, 메모리 사용량이 적어, 멀티쓰레드(Multi Thread)에 강하다. 또한 C언어 기반으로 개발된 언어라, C언어로 개발하는 프로그램에 스크립트 임베딩에 적합하다[11].

모델과 분리되어 전술의 작성 및 잦은 변경을 처리하는 전술처리기의 특성상, 가볍고 빠른 스크립트 언어/엔진이 요구된다. 또한 C언어로 작성된 모델 및 시뮬레이션 엔진과 효과적으로 연동되기 위해서는 스크

립트 임베딩에 적합해야하며, 군단급 교전 시뮬레이션을 지원하기 위해서는 동시에 많은 모델의 전술을 결정하여야 하며, 이에 적은 메모리 점유와 강한 멀티쓰레드 기능이 요구된다.

이를 위해 사용자 입력 쓰레드, 1부터 100까지의 합을 계산하는 쓰레드, 각기 다른 파일로부터 데이터를 읽어 들인 후, 0.05초의 지연 후 한 문장씩 출력하고, 간단한 if-then-else 연산을 하는 쓰레드 등 4개의 쓰레드로 구성된 멀티쓰레드 프로그램을 Python과 Lua로 각각 구현하였다. 각기 실행 속도를 비교한 결과 Lua가 0.291초, Python이 0.695초로 Lua가 강한 멀티 쓰레드 성능을 보였다(Fig. 9).

• **Lua - 4 multi-thread test**

```
*** 예제[Thread02] 1부터 100까지의 합 예제 ***
SCRIPT[0]: width =1000
SCRIPT[1]: width =200
SCRIPT[2]: width =1000
SCRIPT[0]: height =500
SCRIPT[1]: height =300
SCRIPT[2]: height =500
SCRIPT[0]: result is YES
SCRIPT[1]: result is NO
SCRIPT[2]: result is YES

rec_fact : 290728.255 usec
Press any key to continue
```

• **Python - 4 multi-thread test**

```
*** 예제[Thread02] 1부터 100까지의 합 예제 ***
SCRIPT[0]: width =1000
SCRIPT[1]: width =200
SCRIPT[2]: width =1000
SCRIPT[0]: height =500
SCRIPT[1]: height =300
SCRIPT[2]: height =500
SCRIPT[0]: result is YES
SCRIPT[1]: result is NO
[7648 refs]
SCRIPT[2]: result is YES

rec_fact : 694769.996 usec
Press any key to continue
```

Fig. 9. Python, Lua 멀티쓰레드 실행속도 비교

TMA 시뮬레이션을 표현한 정술정의 파일을 각각 Python과 Lua로 각각 구현하고, 이를 처리할 수 있는 각각의 스크립트 전술처리기를 구현한 후, 동일한 상태조건의 전술을 결정하는데 소요되는 시간을 Fig. 10과 같이 비교하였다. Lua 전술처리기가 0.004초, Python 전술처리기가 0.06초의 속도를 나타내어, Lua가 15배 빠른 속도를 보여주고 있다.

```
Script-based Tactic Manager(lua)
 Tactic filename decided to be opened is 'approach.lua'
 generated random number is 0.909091
 Tactic decided by tactic manager is 'LAG'

rec_fact : 4049.855 usec
Script-based Tactic Manager(python)
[7680 refs]
 Tactic filename decided to be opened is 'approach.py'
 generated random number is 0.909091
[8034 refs]
 Tactic decided by tactic manager is 'LAG'
rec_fact : 61383.789 usec
Press any key to continue
```

Fig. 10. Python, Lua 전술처리기 실행속도 비교

일반적으로 효과도 분석을 위해서는 조건을 여러번 달리하여, 각기 1000번 이상의 시뮬레이션을 통해 통계를 얻어내는 Monte-Carlo 기법이 활용된다[7]. 이 때, 하나의 전술만을 결정하는 간단한 시나리오라고 해도, Lua 전술처리기는 4초가 소요되고, Python 전술처리기는 60초가 소요된다. 전술을 결정해야할 시뮬레이션 모델이 많아지고, 한 모델이라도 상황에 따라 다양한 전술을 결정하는 복잡한 교전 시뮬레이션 경우에는 Lua 전술처리기가 큰 이점을 보인다. 즉, 멀티 쓰레드 측면이나, 실행 속도 측면에서 Lua가 Python보다 전술처리기에 더 적합하다.

바. GUI기반 전술편집기 구현

전술의 입력, 수정, 저장, 재사용 등을 전술정의파일의 형식과 관계없이 사용자가 쉽게 작성할 수 있도록, GUI기반의 전술편집기를 Fig. 11과 같이 개발하였다. 전술편집기는 테이블 형식, Python기반의 스크립트 형식, Lua기반의 스크립트 형식 등, 사용자가 입력하고자 하는 전술정의파일의 종류를 선택할 수 있다. 또한

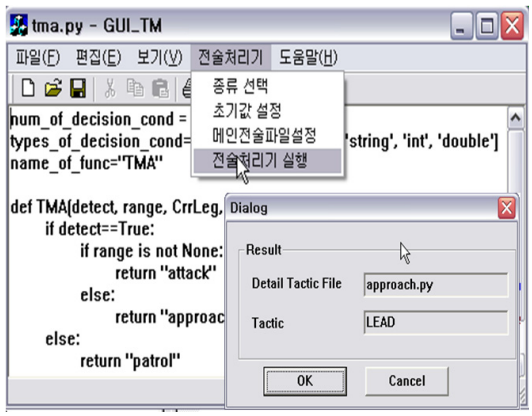


Fig. 11. GUI기반의 전술편집기

모델로부터 받는 속성값을 '초기값 설정'기능을 통해 설정하고, '전술처리기 실행'기능을 통해, 결정된 전술을 얻을 수 있어, 사용자가 본인이 작성한 전술을 검증할 수 있다. 이를 통해 모델링이나 시뮬레이션의 오류가 아닌, 전술입력의 오류로 발생되는 엉뚱한 시뮬레이션 결과를 방지할 수 있게 된다.

3. Case Study

스크립트 형식의 전술처리기가 교전급 시뮬레이션에서 실시간 전술결정에 적합한 지를 검증하기위해, 잠수함 대 수상함 교전시의 표적기동분석(TMA) 시뮬레이션의 전술 결정에 스크립트 전술처리기를 활용하였다. 선행 연구[4]에서는 같은 시뮬레이션에 대해 테이블 형식의 전술처리기를 이용하여 실시간 전술처리를 하였으며, 본 연구는 동일한 전술결정 성능을 보장하기 위해, 테이블 형식의 전술정의 내용과 동일한 전술을 Python과 Lua로 각각 구현하여, 시뮬레이션을 진행하였다.

가. 잠수함 대 수상함 교전시의 표적기동분석(TMA) 시뮬레이션

시뮬레이션의 대상으로 임무를 위해 고속으로 목표를 향해 항주하는 수상함과 이를 발견하고 저지하는 잠수함과의 교전 시뮬레이션을 다룬다. 특히 관심있게 관찰하는 대상은 잠수함이며, 잠수함은 적의 방사소음만을 이용하여 적의 위치, 속도 등을 추정하는 표적기동분석을 수행한다[8,13]. 표적기동분석은 크게 세 번의 기동으로 이루어지며 마지막 기동이 종료되면 목표 함정과의 거리, 위치, 함정의 속도와 진행방향을 알 수 있고 이때의 함정의 위치와 잠수함의 위치를 이용하여 잠수함이 접근하여 목표 함정을 공격할 수 있는 지 여부를 판단한다.

본 논문에서 다루는 표적기동분석은 Fig. 12에 설명된 바와 같이, 방위율 만을 사용하는 표적기동분석(Bearing only TMA)으로, 적함이 변침하지 않고 일정한 속력으로 기동하고 있을 때, 일정시간 동안 일정한 속력으로 기동하는 아 잠수함이 적의 방위(Bearing)과 방위변화율(Bearing Rate)를 측정하고, 변침후 다시 방위율을 측정하여, 보외법(Extrapolation)으로 적의 위치를 추정하는 방법이다. 표적기동분석 원리와 자세한 시나리오는 선행연구를 따른다[4,5].

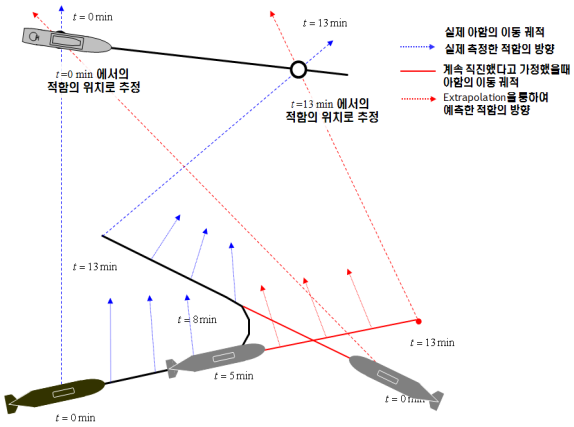
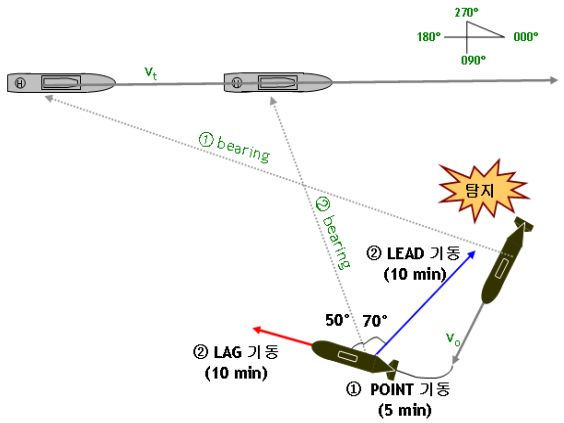


Fig. 12. 방위율을 이용한 표적기동분석



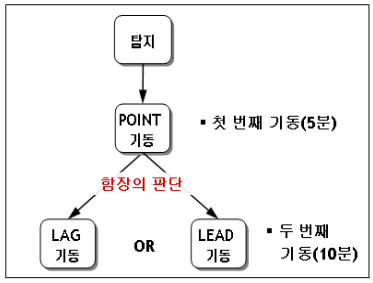
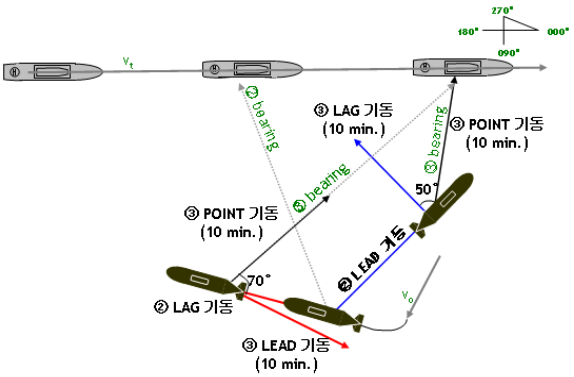


Fig. 13. 표적기동분석의 첫 두 기동

표적기동분석을 위한 기동순서를 결정하는 전술은 다음과 같다. 잠수함이 적 수상함을 탐지하면 5분간 적의 방위로 직진하는 POINT 기동을 수행한다. 이를 통하여 적 수상함의 방위변화율(Bearing Rate)이 분석되면 두 번째 기동을 수행한다. 두 번째 기동은 10분간 진행되며 방위의 변화율 방향으로 직진하는 LEAD 기동과 방위의 변화율 방향 반대방향으로 움직이는

LAG 기동 중 하나로 결정한다(Fig. 13). 두 번째 기동이 끝나면 POINT 기동을 수행하거나 두 번째 기동과 다른 기동을 선택한다. 즉, 두 번째 기동이 LEAD 기동이었다면 LAG 기동이, LAG 기동이었다면 LEAD 기동이 선택된다(Fig. 14).



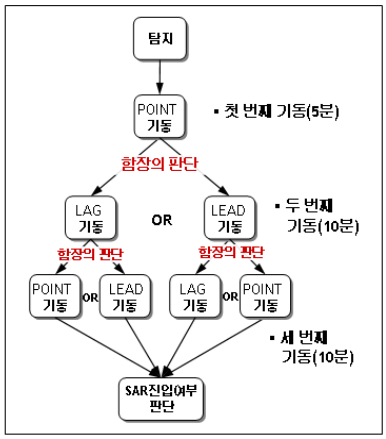


Fig. 14. 표적기동분석의 최종 기동

나. 시뮬레이션 결과

본 연구에서는 이산사건 시뮬레이션을 위한 DEVS 형식론을 이용하여 표적기동분석 시뮬레이션 모델을 구현하였고 여기에 전술처리기와 모델과의 인터페이스를 개발하여 적용하였다. DEVS 모델구조에 따른 계층적 모델구조[15]를 지원하는 시뮬레이션 엔진[3]을 사용하여 시뮬레이션을 구성하였다. Fig. 15는 개발된 표적기동분석 시뮬레이션 프로그램의 실행화면을 보여주고 있다.

본 시뮬레이션을 수행하면, 외부 사건(Event)가 발생할 때마다, 전술처리기가 호출되어 모델에 전술을 전달하게 된다. 탐지 및 접근 단계의 예를 들면, 표적기동을 완수하기 위해서, 모델은 3번의 다른 형태의 기동을 결정하여야 하고, 이 때 마다 전술처리기가 호출된다. Fig. 15에서 알 수 있는 것과 같이, 붉은선으로 표시된 잠수함의 이동 궤적은 지그재그형으로 각기 다른 3번의 기동이 수행되어 표적기동분석을 완료하였음을 알 수 있다.

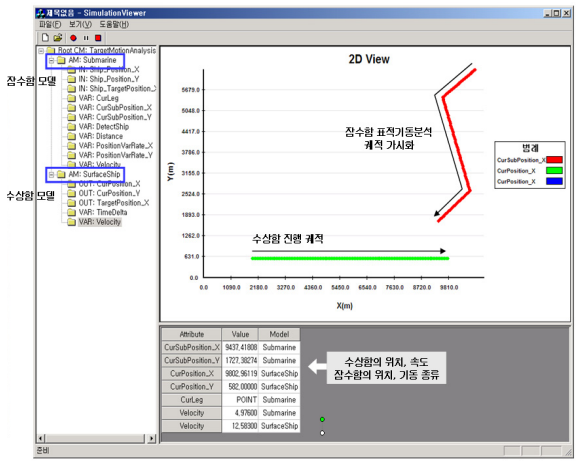


Fig. 15. 표적기동분석 시뮬레이션 실행화면

개발된 프로그램을 이용하여 아 잠수함의 속도를 2~8knots, 적 수상함의 속도를 10~24knots로 변경시켜가면서 표적기동분석 시뮬레이션을 수행하고 적 수상함에 대한 공격 성공확률을 분석하였다(Fig. 16). 분석 결과, 아 잠수함의 속도가 빨라질수록 공격 성공확률이 증가하며, 적 수상함의 속도가 빨라질수록 공격 성공 확률이 감소하였다. 재래식 잠수함의 평균 작전 운용 속도인 2~4knots에서는 적 수상함의 속도가 14knot일 때, 최고의 공격 성공확률을 보인다.

다음으로, 전술정의 파일에서 표적기동분석을 위한 네 가지 세부 기동으로 분기하는 확률을 제거하여, POINT-LEAD-POINT, POINT-LEAD-LEG, POINT-LAG-LEAD, POINT-LAG-POINT와 같이 총 4가지 기동순서를 결정론적인 방법으로 시뮬레이션을 각각 수행하여 공격 성공 확률을 분석해보았다. 아 잠수함의 속도를 2~8knots로 균일하게 분포시키고, 적 수상함의 속도를 10~24knots로 변경시켜가며, 네 가지 서로 다른 TMA 기동 전술을 각각 시뮬레이션하였다. Fig. 17은 표적기동분석을 위한 기동순서로 POINT-LEAD-POINT를 택했을 경우가 공격성공 확률이 가장 높음을 보여주고 있다.

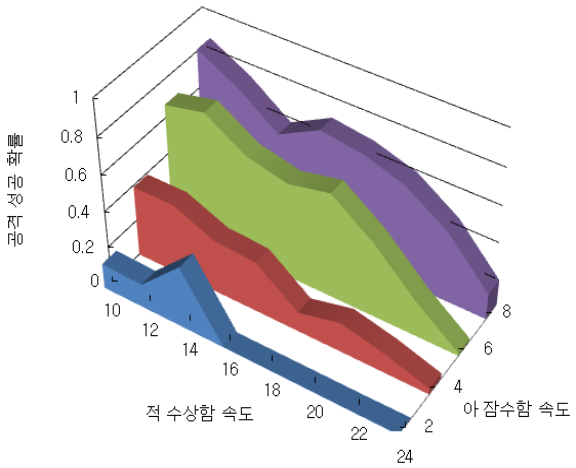


Fig. 16. 수상함, 잠수함의 속도에 따른 공격 성공 확률

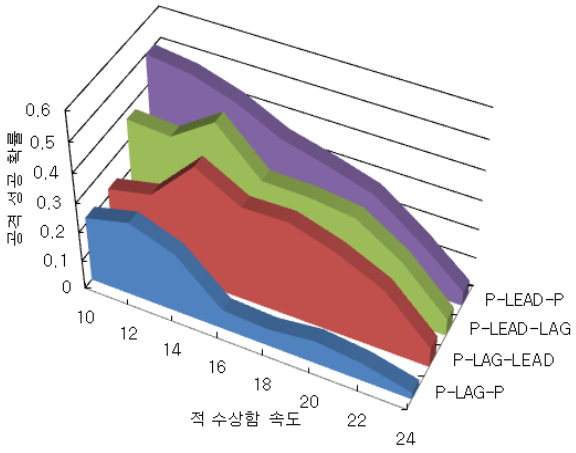


Fig. 17. TMA 기동 순서에 따른 공격 성공 확률

```
Table-based Tactic Manager
 Tactic filename decided to be opened is 'approach.txt'
 generated random number is = 0.909091
 Tactic decided by tactic manager is 'LAG'

execution time : 2589.692 usec
Script-based Tactic Manager(lua)
 Tactic filename decided to be opened is 'approach.lua'
 generated random number is 0.909091
 Tactic decided by tactic manager is 'LAG'

execution time : 1870.063 usec
Script-based Tactic Manager(python)
[7680 refs]
 Tactic filename decided to be opened is 'approach.py'
 generated random number is 0.909091
[8034 refs]
 Tactic decided by tactic manager is 'LAG'
execution time : 54057.229 usec
Press any key to continue
```

Fig. 18. 선행연구[4]와 본 연구 전술처리기 속도 비교

Fig. 18은 표적기동분석 시뮬레이션을 진행하면서 사용된 전술처리기의 한 개의 전술 결정 속도를 선행연구[4]와 본 연구로 구분하여 나타내고 있다. 선행연구에서 사용된 테이블 전술처리기는 표적기동분석의 기동에 대한 전술을 하나 결정하는데 0.0026초가 소요되는 반면, 본 연구에 사용되는 스크립트 형식의 전술처리기 중, Lua는 0.0019초, Python은 0.054초가 소요되었다. '2-마'절에서 분석하였듯, 스크립트 전술처리기는 Lua가 Python보다 더 적합하며, Fig. 18에서 보는 것과 같이 테이블 형식의 전술처리기보다 Lua 스크립트 전술처리기가 더욱 빠른 실행속도를 보인다. 실행속도 뿐만 아니라, '2-라'에서 분석한 바와 같이, 스크립트 전술처리기는 복잡한 전술 표현과 뛰어난 가독성, 전술정의의 편의성 등 다양한 장점이 있어, Lua 스크립트 전술처리기가 교전급 시뮬레이션의 전술처리기로 적합하다.

본 연구에서 제안한 전술처리기의 개념을 이용하여 다양한 전술에 대한 시뮬레이션을 쉽고 빠르게 수행한다면 실전에서의 잠수함 표적기동분석을 위한 전술결정에 대한 노하우를 축적할 수 있을 것이다.

4. 결론 및 향후 연구계획

본 연구에서는 시뮬레이션을 통하여 다양한 전술의 효과를 예측할 수 있도록 고안된, 모델과 분리된 전술처리기를 효과적으로 사용하기 위해, 스크립트 기반 전술처리기를 연구하였다. 전술정의 파일을 쉽게 작성할 수 있고, 다양한 전술을 표현할 수 있는 스크립트 언어를 이용한 전술정의법을 채택하고, 이를 실행할 스크립트 기반 전술처리기를 구현하였다. 모델과 전술처리기 사이의 인터페이스, 전술처리기와 스크립트 전술정의파일간의 인터페이스에 대해 분석하고, 이종의 대표적인 스크립트 언어에 대해서 전술처리기 관점에서 비교 분석하였다. 구현된 전술처리기(Python, Lua)를 이용하여, 테이블 기반 전술처리기와 동일한 방식으로 표적기동분석 시뮬레이션을 수행하였고, 기존연구[4]와 동일한 결과를 얻었으며, 실행 속도는 Lua 스크립트 전술처리기의 경우 더욱 빨랐다. 실행속도와 멀티쓰레딩능력, 전술표현의 편의성, 가독성, 다양성 등의 측면에서 Lua기반 스크립트 전술처리기가 교전급 시뮬레이션의 전술처리기로 우수한 성능을 보였다.

향후 연구계획으로는 멀티쓰레딩 환경에서 다양한

전투객체의 전술을 실시간으로 처리하는 연구를 진행할 것이다. 또한, 확률에 의한 전술 처리 대신 인공지능 중 Fuzzy 기법을 활용하여 전술을 표현하는 연구를 수행중이다.

후 기

본 연구는 (1) 국방과학연구소 수중운동체기술특화연구센터 SM-11과제 “수중 운동체의 체계/부체계 기능 및 성능 시뮬레이션을 위한 네트워크 기반의 가상(Virtual) 복합 시스템 모델 구조(Architecture) 연구”, (2) 서울대학교 BK 21 해양기술인력양성사업단의 지원으로 이루어진 연구 결과의 일부임을 밝히며, 이에 감사드립니다.

Reference

[1] 노명일, 이규열, “객체 지향 초기 선체 구조 설계 시스템 개발”, 한국 CAD/CAM 학회 논문집, Vol. 10, No. 4, pp. 244～253, 2005.

[2] 박준규, “개념설계 단계의 잠수함 작전효과도 시뮬레이션 모델 연구”, 한국군사과학기술학회지, Vol. 7, No. 3, pp. 47～58, 2004.

[3] 방경운, 조선공정 계획용 이산 사건과 이산 시간 혼합형 시뮬레이션 프레임워크, 석사학위논문, 서울대학교, 2006.

[4] 조두연, 손명조, 차주환, 이규열, 김태완, 고용석, “잠수함의 표적기동분석 시뮬레이션을 위한 전술처리기의 구현”, 한국시뮬레이션학회논문지, Vol. 16, No. 3, pp. 65～74, 2007.

[5] Bakos, G. K., Submarine Approach and Attack Tactics - Simulation and Analysis, Master’s Thesis, Naval Postgraduate School, Monterey California, 1995.

[6] Buckland, M., Programming Game AI by Example, World Publishing, Inc., 2004.

[7] Cho, D. Y., Son, M. J., Kang, J. H., Lee, S. J., Cha, J. H., Yoo, S. J., Lee, H. K., Lee, K. Y., Kim, T. W., Ko, Y. S., “Analysis of a Submarine’s Evasive Capability Against an Antisubmarine Warfare Torpedo using DEVS Modeling and Simulation”, Spring Simulation Multiconference 2007, DEVS Integrative M&S Symposium (DEVS), Norfolk Marriott Waterside, Norfolk, Virginia, USA, Mar. 25～29, 2007.

[8] Cunningham, A. and Thomas, A., “Target Motion Analysis Visualization”, Asia Pacific Symposium on Information Visualisation (APVIS 2005), Sydney, Australia, Conference in Research and Practice in Information Technology, Vol. 45, 2005.

[9] Jung, K., Brown A., Beginning Lua Programing, Wiley Publishing, Inc., 2006.

[10] Ierusalimschy, R., Programming in Lua, 2nd ed, Lua.org, 2006.

[11] Lua official webpage: http://www.lua.org/

[12] Lutz, M., Programming Python, 3rd ed., O'Reilly Media, Inc., 2006.

[13] Nardone, S. C. and Graham, M. L., “A Closed-Form Solution to Bearings-Only Target Motion Analysis”, IEEE Journal of Oceanic Engineering, Vol. 22, No. 1, pp. 1～11, 1997.

[14] Varanese, A., Game Scripting Mastery, Course Technology PTR, 2002.

[15] Zeigler, B. P., Object-Oriented Simulation with Hierarchical, Modular Models, Academic Press, 1990.

[16] Zeigler, B. P, Praehofer, H. and Kim, T. G., Theory of Modeling and Simulation, 2nd ed., Academic Press, 2000.