2015 춘계공동학술대회 KORMS/KIIE/ESK/KSIE/KSS
일시: 2015.4.8(수)~4.11(토) / 장소: 라마다프라자 제주호텔

C-Interpreter를 이용한 유연한 시뮬레이션 초기화 방법에 관한 연구
Research on Flexible Method for Simulation Initialization using C-Interpreter

신용준[1], 양지용[1], 최창범[2]
[1]한동대학교 전산전자공학부, [2]한동대학교 창의융합교육원
[1]{21300408, 21000416}@handong.edu, [2]cbchoi@handong.edu

Abstract

In order to develop a proper simulation model, a developer should define the requirements which complies with the simulation objectives, and analyze the requirements to design the simulation model. Then, the developer may implement the simulation model and finalize the simulator. However, the requirement specification of a simulation model may frequently change during the development process. As a result, initial information of the simulation model may change due to the specification changes, so the developer should modify the initialization modules and its simulation models. Also, developing simulator based on design of experiments, developer initialize factors that be experimented. However, there can be many experimented factors affecting results, and it is hard to manually initialize all these factors. Such tasks are tedious job and software defects may be included to the simulator during the modification. To tackle this problem, we propose C-interpreter based flexible initialization method to give initial information to the simulation models based on C-programming language. The proposed method is distinguished from existing initialization methods, such as XML files or legacy scenario files. Finally, we apply the initialization method to DEVSim++ based simulator to show the effectiveness.

1. 서론

시뮬레이션은 상업, 공학, 국방 등 많은 분야에서 사용되며, 사용자의 목적에 따라 현실의 다양한 상황을 모의 할 수 있다는 점에 있어 많은 분야에서 활용되고 있다[1]. 시뮬레이션은 현실 세계를 반영하여 모델링을 수행한 후 시뮬레이션 하고자 하는 가정에 해당하는 시나리오를 해당 모델에 적용, 실행함으로써 의사결정 등에 대한 통찰력을 얻을 수 있으며 대표적인 사례로 국방의 워 게임 시뮬레이터를 들 수 있다. 워 게임 시뮬레이터는 적은 비용으로 많은 상황에 대하여 실험을 수행하고 모의 전략을 수립하여 실 상황에 적절한 대비가 가능하게 한다. 실제로 현 국방에서는 많은 부분에서 모의 전략 수립을 위하여 워 게임 시뮬레이터를 사용하고 있다.

시뮬레이션 기반의 실험 데이터를 통하여 현실 세계에 대한 통찰력을 얻기 위해서는, 정확한 시뮬레이터 개발 및 시나리오 작성이 필요하다. 이때 시뮬레이터의 개발에 있어, 시뮬레이션 목적에 맞는 요구사항을 정립하고 그 요구사항에 맞는 모델을 설계해야 한다. 이후 각 모델들이 요구사항을 올바르게 반영하는지 검토 한 후, 세부 모델들을 각각 구현하고 통합하여 시뮬레이터를 완성한다. 각 모델들은 요구사항을 적절히 표현하기 위해 고유의 변수, 모델명 및 상태들로 표현되며, 이것들의 초기화 과정을 통해 시뮬레이션 환경이 초기화된다. 이상적인 경우 시뮬레이터 개발과정에서 결정한 초기화 정보는 시뮬레이터 활용 전까지는 변경되지 않는다. 그러나 모델을 개발하는 과정 중에 시뮬레이션의 목적이 변경되거나, 모델 및 시뮬레이션 환경 값들의 요구사항이 변경되는 경우가 빈번하게 발생한다. 한 번 결정된 초기화 구조를 변경하는 것은 초기화 파일 및 시뮬레이터 코드에 수정이 필요하므로 변경이 어렵다. 이러한 경우 요구사항을 표현한 고유의 변수 및 상태들의 값을 포함한 새로운 구조에 대해 다시 명시해야한다. 이는 해당 요소들에 대한 초기화 루틴 또한 수정해야하므로 이러한 상황에 대한 효율적인 해결 방법에 대한 연구가 필요하다.

또한, 시뮬레이션 결과를 효율적으로 획득하기 위하여 개발된 시뮬레이터에 실험 계획법을 적용하여 시뮬레이션 결과를 획득 할 수 있다. 실험계획법이란, 실험하고자 하는 독립변수들에 대한 종속변수와의 관계를 알고자 할 때 사용되며, 여러 연구 분야에 폭넓게 사용되는

방법이다. 실험계획법은 여러 개의 독립변수들을 가질 수 있으므로 시뮬레이터의 개발 완료 이후, 독립변수들에 대한 초기화 정보를 변경하며 반복적으로 시뮬레이션을 수행한다. 즉, 실험계획법에 기반 한 시뮬레이션이 필요한 경우, 실험계획법에 의거한 다수의 시나리오 생성 및 관리가 필요하다. 그러나 현재 사용되고 있는 시나리오 파일을 이용한 초기화는 직접 시나리오 파일 내의 모든 변수를 수정해야 한다는 점에서 수동적이며 실험계획법의 반복적인 초기화 루틴을 지원하기에 부적합하다.

따라서 일반적인 시뮬레이터 개발 과정에서 초기화 구조 변경 지원과 시뮬레이터에 실험계획법을 적용하는 경우 유연하고 동적인 초기화 방법에 대한 연구가 필요하다. 본 논문은 C/C++ 스타일 기반의 시뮬레이션 구현에 있어서 C-interpreter를 이용한 유연한 초기화 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 시뮬레이터 초기화와 실험계획법에 대한 배경지식을 소개한다. 3장에서는 C-interpreter를 이용한 유연한 초기화 방법을 제안하고 4장에서 해당 초기화 방법을 적용하여 사례연구를 보인다. 마지막으로 5장에서는 결론을 맺는다.

2. 배경지식

2.1 시뮬레이터 초기화 구조

시뮬레이터의 초기화 구조는 시나리오 파일을 이용하며, 이를 해석하여 해당 변수, 모델명 및 상황들에 해당하는 값들의 초기화를 수행한다. 시나리오 파일은 XML(eXtensible Mark up Language), INI(Initialization), 혹은 TXT(Text) 파일 등으로 표현되며 정적인 구조를 가진다. 사용자는 시나리오 파일에 변수들의 초기 설정에 필요한 값들을 기입해두고, 시뮬레이터가 이를 해석해 초기화 하는 방식을 가진다. 이러한 시나리오 파일의 사용은 코드에 직접 수정을 가하지 않기 때문 시뮬레이션 모델의 재사용 할 수 있어 개발 비용을 낮춘다. 그러나 시나리오 파일은 시뮬레이션의 목적의 변경이나, 시나리오의 변경에 있어 수동적인 변경을 해야 한다. 또한 시나리오 파일은 정적인 표기 방식이기 때문에 초기화 구조의 추가나 삭제에 있어 유연하지 못하다는 단점이 있다.

2.3 실험계획법

실험계획법이란 입력에 따른 체계의 출력값을 분석하기 위해 독립변수와 종속변수 간의 관계를 파악하는 방법이다. 반복적인 실험을 통해, 각 독립변수들에 변화를 주며 종속변수에 미치는 영향을 통계적으로 분석한다.

따라서 시뮬레이션에 실험계획법의 적용은 독립변수들의 다양한 초기화를 필요로 한다. 독립변수와 종속변수의 상관관계를 찾기 위해 독립변수들의 각 상황에 따른 다양한 초기화를 통해 종속변수에 미치는 영향을 찾는다. 시뮬레이션 모델들의 변수 및 시뮬레이션 환경 값들을 독립변수로 하여 해당 독립변수들이 종속변수 및 시뮬레이션 결과에 미치는 영향을 분석한다.

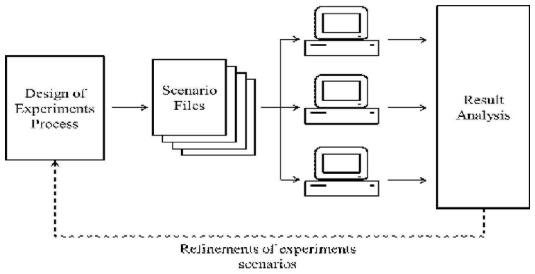

그림 1. 실험계획법에 따른 과정

그림 1과 같이 실험계획법의 과정은 먼저 실험의 목적에 맞는 독립변수들을 정확히 명시하고, 각 변수들에 대한 실험을 계획하여 실험 과정을 설계하고, 해당 독립변수들에 대한 값들을 담은 시나리오 파일을 만든다. 그리고 해당 시나리오를 사용해 실험한 후 결과에 대해 분석하고 그 분석은 다시 새로운 시나리오의 생성으로 환원된다. 기존의 시뮬레이션 초기화 방식을 사용한 실험계획법은 시나리오 파일을 여러 개 만들거나 각각의 케이스에 대해 실험 시 마다 초기화 값을 수정해 시뮬레이션 한다. 독립변수들은 시나리오 파일 안에 명시되어 실험에 맞게 새로운 초기화를 반복적으로 수행하여 실험된다. 그러나 이 방법은 수동적이고 정적이기 때문에 실험계획법을 적용하기에 원활한 환경을 제공하지 못한다.

예를 들어, 실험계획법 중 완전요인배치법(full factorial design)이 있다. 이는 독립변수들에 대해 가능한 모든 조합을 테스트 하는 방식으로 모든 가능성의 결과를 확인할 수 있다. 그러나 완전요인배치법을 시나리오 파일을 사용한 시뮬레이션에 적용 할 경우 모든 조합의 시나리오 파일을 만들거나 매번 수정해야하며 다수의 독립변수들을 수동적으로 초기화하기에 어려움이 있다. 그러나 반복문을 사용한 초기화가 가능하다면 시나리오 파일을 이용한 초기화보다 효율적인 초기화가 가능하며, 시뮬레이션에 실험계획법의 적용을 원활이 지원할 수 있다.

3. 본론

시뮬레이션의 초기화 과정에서 시나리오 파일을 이용하는 방법과 같은 수동적이며 정적

인 초기화는 시뮬레이션 목적의 변경으로 인한 초기화의 구조 변경 및 실험계획법의 적용에 의한 반복적인 재초기화를 지원하기에 적합하지 않다. 이러한 문제의 해결 방안으로 C-interpreter를 사용해 C 스크립트를 사용한 유동적인 시뮬레이션의 초기화 과정을 가능하게 한다.

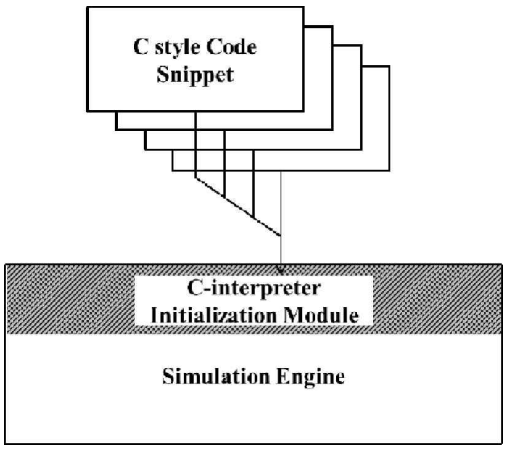

그림 2. C-interpreter를 통한 유연한 초기화 지원 개념도

C-interpreter를 이용한 초기화 방법에 대한 개념도는 그림 2와 같다. 시뮬레이션 엔진의 초기화 부분을 C-interpreter 모듈을 사용해 구현한다. C-interpreter는 C 스타일의 코드를 실행시키는 환경을 제공하며 본 논문은 CERN의 C-interpreter(CINT)를 사용한다. 초기화 모듈에서 C 스타일의 스크립트를 C-intrpreter를 통해 실행하며, 초기화 과정이 진행된다.

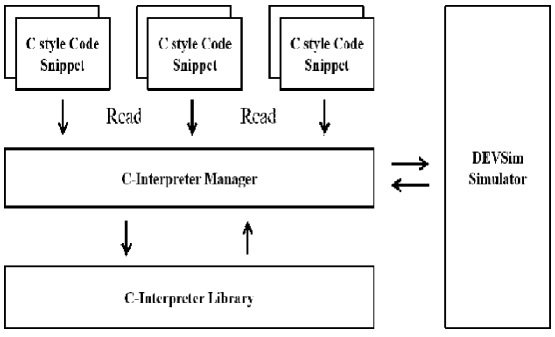

그림 3. C-interpreter를 이용한 초기화 과정

그림 3은 C-interpreter를 이용한 초기화 과정을 보여준다. C-interpreter Manager는 초기화 과정을 담은 C 스타일의 코드를 읽어 C-interpreter Library에 실행을 요청한다. C-interpreter Library는 초기화 코드를 실행하고 초기화를 마친 데이터를 C-interpreter Manager를 통해 시뮬레이터에 전달한다. 전달받은 데이터는 시뮬레이터의 초기화 모듈에서 초기화에 사용된다.

C-Interpreter를 사용하여 시뮬레이션의 초기화를 진행할 경우 장점은 초기화 과정에 있어 C 스크립트를 사용할 수 있다는 것으로 기존의 시나리오 파일을 이용하여 초기화를 하던 방법보다 C 문법을 사용한 유연한 초기화가 가능하다. 또한 반복적인 초기화에 있어 수동으로 시나리오 파일을 수정해야 했던 것을 C 문법을 사용하여 자동적으로 초기화 할 수 있다. 그리고 C의 문법 및 기본 라이브러리를 사용하여 C 프로그래밍 언어가 익숙한 사용자에게 친숙한 초기화를 제공한다. C 스크립트를 이용하여 초기화 할 수 있다는 것은 기존의 초기화 과정처럼 값만을 전달하는 것이 아니라 초기화 값이 나오게 된 과정을 스크립트로 표현할 수 있다는 의의가 있다.

3.1 초기화 구조 변경의 유연성

시뮬레이션의 목적이 변경되거나 각 모델들의 요구사항이 변경 될 때, 초기화 구조의 변경을 필요로 하는 경우가 있다. 예를 들어, 구조체의 변수가 추가되어야 하는 상황에 기존의 시나리오 파일을 사용한 초기화의 경우 시나리오 파일을 새로 작성하거나, 기존의 시나리오 파일을 수정하여 추가된 구조를 기술해야 한다. 그리고 시뮬레이터에서 시나리오 파일을 해석하는 부분 역시 수정이 되어야 할 것이다. 그러나 C 스크립트를 사용하여 초기화 한다면, 구조체에 해당 변수만 추가하여 초기화 모듈에 전송한다. 또한 구조체의 변수가 삭제되어야 하는 경우, 시나리오 파일을 이용한 초기화의 경우 시나리오 파일을 정적으로 수정하고, 시뮬레이터의 이를 해석하는 부분 역시 다시 수정해야한다. 그러나 C 스크립트를 이용한다면 해당 변수를 주석처리하고, 초기화 모듈에 전송한다. 그리고 이것은 버려지지 않고, 주석으로 남기 때문에 재사용에 용이하다.

3.2 실험계획법에 기반 한 초기화의 유연성

다양한 독립변수들과 종속변수들 간의 상관관계를 시뮬레이션 하거나 각 독립변수들 간의 상호작용에 대해 테스트하기 위해서는 독립변수들의 초기화를 제어하기 위한 환경이 필요하다. 이를 위해 C 스크립트를 사용한 초기화를 수행한다. 우선적으로 반복문의 사용이 실험계획법을 뒷받침하기에 적합하다. 예를 들어 완전요인배치법의 경우, 사용자는 모든 가능성에 대해 초기화해야 할 필요가 있으며, 시뮬레이터에서 완전요인배치법을 지원하지 않는다면 직접 반복적인 초기화를 수행해야한다. 그러나 예전의 방식과 달리 반복문을 사용한 초기화가 가능해지면 모든 독립변수들을 일일이 초기화 할 필요가 없어진다. 또한 switch 등의 제어문을 사용해 필요한 케이스에 대한 초기화 과정을 명시해놓고 흐름을 쉽게 관리 할 수 있다.

그리고 독립변수들에 대한 제한사항에 대해 명시하여 관리할 수 있고, 예외적인 초기화의 발생을 막을 수 있다. 이 외에도, 다양한 산술문과 함수를 사용해 필요에 따른 다양하고 복잡한 초기화를 할 수 있고, 라이브러리를 사용한 초기화를 통해 실험계획법을 더 유연하게 지원한다.

3.3 C 문법을 이용한 유연한 초기화의 예시

시나리오 파일이 C 코드 파일로 대체되면서 초기화에 사용 가능한 변수의 타입을 다양하게 해준다. 그 예로 불리안 타입의 변수를 유연하게 초기화 할 수 있다. 시나리오 파일을 사용 할 경우, 변수에 수동으로 값을 기입해 주어야 했지만, C 문법을 통해 참/거짓 값을 반환하는 연산을 자동화 시켜 초기화에 유연함을 더한다. 또한 구조체를 사용해 변수들을 쉽게 관리하며 간단한 초기화 과정을 지원한다.

```
x = y = z = 0;
```

그림 4. 다변수의 초기화 예시

또한 그림 4의 코드처럼 대입 연산자들을 통해 많은 변수들이 항상 동일한 값을 가져야 하는 경우에 쉽게 초기화 할 수 있고, 변수들의 관계를 정확히 명시해서 초기화 과정에서의 오류를 줄일 수 있다.

```
int x = rand() % 10;
int y = (x * x) + (2 * x) + 5;
int z = cos(y)
```

그림 5. 함수를 사용한 초기화 예시

또한 그림 5의 코드처럼 함수 및 수식을 이용하여 간편한 초기화를 진행할 수 있다. 수식 연산을 통해 초기화하는 변수는 어떠한 인자의 전달에도 동일한 과정을 거친 값으로 초기화 할 수 있다. 그리고 C-interpreter를 통해 C의 표준 라이브러리를 사용한 초기화가 가능하다. math.h를 사용하여 기본적인 수학 함수를 제공 받을 수 있으며, random.h를 사용하여 난수를 사용한 초기화도 가능하다. 이것 외에 다양한 표준 라이브러리를 사용해 다양하고 유연한 초기화가 가능해진다.

그리고 반복문을 이용하면 연산을 간소화고 코드의 흐름을 반복해 유연한 초기화를 할 수 있으며 배열에 대한 초기화가 가능하다. C 문법을 사용하여 배열의 초기화가 가능하기 때문에 규모가 큰 시뮬레이터의 초기화에 효율을 더한다. 또한 배열의 요소들 간의 관계를 파악하기에 더 효율적이다.

4. 사례 연구

C-interpreter를 이용한 유연한 초기화의 적용을 위해 이산 사건 시스템 형식론의 DEVSim++를 사용한 시뮬레이터를 구현하여 유연한 초기화를 수행한다.

4.1 이산 사건 시스템 형식론

본 논문에서는 C++기반의 DEVSim++로 시뮬레이터를 구현하여 C-Interpreter를 이용하여 유연한 초기화를 진행한다. DEVSim++의 이산사건 시스템 명세 (Discrete Event System Specification : DEVS)형식론은 집합론에 근거한 시뮬레이터 개발 도구로서, 객체 지향 모델링의 특징을 지닌다. DEVS 형식론을 통하여서 시뮬레이터를 개발할 때는 이산 사건에 대하여 객체 단위의 원자모델을 만들고, 각 모델들을 결합(결합모델)하여 모델간의 상호작용 및 계층적 구조를 구성한다. 또한 DEVS 형식론은 모델의 행동 및 상황을 표현하는 특징을 가지고 있어서 실제 상황을 반영하는 시뮬레이션을 수행할 수 있는 장점이 있다. 다음으로 DEVS 형식론의 원자모델과 결합모델에 대해 설명한다.

4.1.1 원자모델

원자 모델은 DEVS 형식론을 구성하는 가장 기본적인 모듈로써 원자 단위의 분할을 통하여서 시스템의 행동을 기술하는 모델이다. 원자 모델 M의 수학적인 표현은 그림 6과 같다.

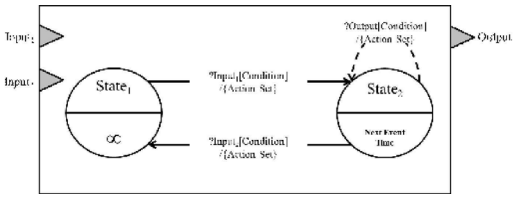

그림 6. 이산 사건 시스템 형식론의 원자 모델

$AM = \langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$
X : 이산사건 입력 집합
Y : 이산사건 출력 집합
S : 일련의 이산사건 상태의 집합
$\delta_{ext} : Q \times X \rightarrow S$: 외부 상태 천이 함수
$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$
: 시스템 M의 전체 상태
$\delta_{int} : Q \rightarrow Q$: 내부 상태 천이 함수
$\lambda : Q \rightarrow Y$: 출력 함수
$ta : S \rightarrow R^{(0,\infty+)}$: 시간 진행 함수

원자모델은 시스템의 상태에 맞게 S(State)를 정의하고, 외부 입력에 대한 변화를 나타내는 상태 천이 함수와 시간의 흐름에 따른 변화

를 나타내는 내부 천이 함수를 구분하여서 정의하고, 각 상태의 변화에 따라 모델이 실행해야하는 조건부 행동을 명시하도록 한다.

원자모델에서는 이산사건 입력 집합, 일련의 이산사건 상태의 집합, 시간 진행 함수 및 사용자에 의해 정의 되었거나 미리 선언된 기본형에 초기화를 진행한다.

4.1.2 결합모델

결합모델(Coupled Model)은 원자 모델 및 결합 모델들을 내부적으로 연결하여 만든 모델로 계층적 구조를 형성하여 더 큰 시스템을 구현할 수 있다.

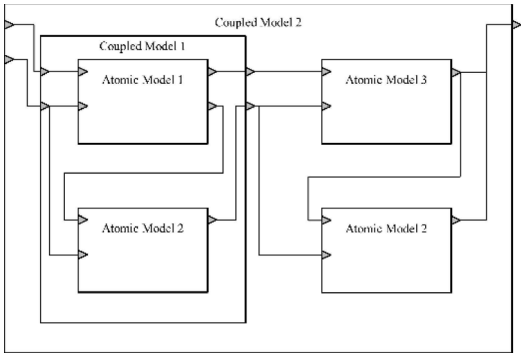

그림 7. DEVS 형식론의 결할 모델

CM = <X, Y, {M_i}, EIC, EOC, IC, SELECT>
X : 이산사건 입력 집합
Y : 이산사건 출력 집합
{M_i} : 모든 이산사건 컴포넌트 모델들의 집합
EIC : 외부 입력 연결 관계
EOC : 외부 출력 연결 관계
IC : 내부 연결 관계
SELECT : $2^{\{Mi\}}- Ø \rightarrow M_i$: 같은 시각에 존재하는 사건을 발생하는 모델들에 대한 선택 함수

그림 7은 결합 모델을 통하여서 DEVS 형식론의 계층적 구조와 재사용성을 보여준다.

결합모델에서의 초기화는 결합모델 내에 속한 사용자 정의 혹은 기본형 변수나, 다른 모델의 계층적 구조를 통한 인자전달로 초기화를 진행한다.

4.2 실험 모델

유연한 초기화 방법의 적용을 위해 DEVSim++를 이용해 간단한 시뮬레이터 모델을 만들어 사용했다. 본 시뮬레이터는 GBP(Generate-Buffer-Process) 구조로 초기화에 대한 단순 동작 검증을 위해 간단한 구조로 디자인했다.

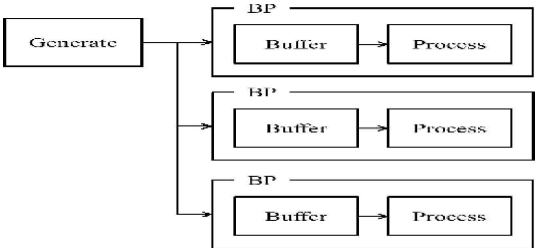

그림 8. GBP(Generate Buffer Process) 구조

그림 8의 생성(Generate) 단계에서는 테스트하고자 하는 객체를 생성하는 과정이다. 버퍼(buffer, 완충) 단계에서는 생성된 객체를 전달 및 보관하는 단계이다. 그리고 가공(Process) 단계는 전달받은 객체를 가공하거나 시뮬레이션 목적에 부합하는 결과를 보여주는 단계이다. 생성-완충-가공의 간단한 세 단계로만 구성된 시뮬레이터에 C-interpreter를 이용해 각 모델의 변수들을 초기화한다.

C-interpreter를 사용한 시뮬레이터 구현에 있어서, 시뮬레이터와 C-interpreter는 별개의 환경에서 실행된다. 초기화 과정을 실행하는 C-interpreter와 초기화를 요구하는 시뮬레이터는 같은 메모리를 참조하지 못하기 때문에 전달하고자 하는 데이터를 C-interpreter의 데이터 전송 함수를 사용하여 전송해야한다. 예를 들어 simulator.h에 init_time이라는 변수를 초기화 하고자 하는 상황을 본다.

```
int init_time;
CintManager manager;
manager.sendFileToCint("initialization.c");
//초기화 과정을 담을 파일을 Cint로 실행
manager.readDataFromCint(&init_time,"init_time
");
//Cint로부터 init_time을 초기화
```

그림 9. 시뮬레이터의 초기화 모듈 simulator.h

```
init_time=100;
```

그림 10. 초기화 파일 initialization.c

그림 10의 초기화 파일은 변수를 어떻게 초기화 할 것인지에 대한 코드를 담은 파일이다. C-interpreter이기 때문에 C 문법을 사용한 초기화가 가능하다. 그림 9의 초기화 모듈 에서는 Cint를 실행시키고 실행한 결과를 가지고 초기화 하고자 하는 데이터를 일대일 대입 시켜주는 과정의 코드를 담는다. Cint를 실행시킬 객체를 생성 후 초기화 과정을 담은 파일(initialization.c)을 실행시키고, 변수에 초기화할 데이터를 넣어준다. 초기화를 위한 가장 간단한 과

정을 예시로 보였으며 C-interpreter를 이용해 어떤 유연한 초기화를 수행하였는지 초기화 코드(initializtion.c)에 주목하여 보인다.

4.3 사례 구현

GBP 구조의 시뮬레이터를 구현하여 시나리오 파일을 사용한 초기화와 C-interpreter를 사용한 유연한 초기화를 비교한다. 대입연산자, 수식사용 그리고 완전요인배치법 부분에서 검증을 진행하였다. 시나리오 파일을 이용한 초기화의 경우, 대입연산자 또는 완전요인배치법에 대한 초기화에 있어서 초기화 할 변수의 값을 모두 직접 수정해야 하며, 수식은 사용자가 직접 계산을 하여 초기화해야 한다. 다음은 본 논문에서 제시하는 바를 적용한 구체적 사례이다.

```
typedef struct {
    int x;  int y;  int z;
} position;  position pos;
pos.x = pos.y = 10;
pos.z = rand() % 100 - 50;
```

그림 11. 대입연산자와 함수를 사용한 초기화

그림 11에서는 x와 y의 좌표 값이 동일한 값을 가질 때 대입연산자를 통하여서 효율적인 초기화가 가능하다. 그리고 z의 값의 경우 random 함수를 통하여서 초기화를 진행하여 일정 분포에서 난수를 통하여 결과를 보고자 할 때 유용하다.

```
pos.z = 0;
pos.x = 3;
pos.y = pos.x * pos.x - 6 * pos.x + 9;
```

그림 12. 수식을 이용한 초기화

그림12에서는 C코드를 이용하여 다항식을 초기화에 이용한다. 기존의 시나리오 파일을 통한 초기화 방식으로는 그림 10과 그림 11의 과정을 일일이 계산 및 수정 하여 초기화 해야 했다. 하지만 본 논문에서 제시한 방법을 사용함으로써 직관적이며 유연한 초기화를 할 수 있다.

그림 13에서는 완전요인배치법과 같이 모든 조합에 대하여 확인하고자 할 때 반복문을 사용한 초기화 과정을 보인다. 기존의 방식이라면 이 모든 가능한 경우를 확인하기 위해서 시나리오 파일을 다 일일이 수정했어야 하나 위 방식을 통하여 효율적인 초기화가 가능하다.

```
int i; int j; position pos[100]
for(i = 0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        pos[10 * i + j].x = i;
        pos[10 * i + j].y = j;
        pos[10 * i + j].z = 0;
    }
}
```

그림 13. 완전요인배치법에 따른 초기화

5. 결론

시뮬레이터의 개발에 있어서, 시뮬레이션 목적에 맞는 시뮬레이션 모델들의 초기화 과정이 필요하다. 이러한 초기화 과정을 C-interpreter를 이용하여 유연하게 수행할 수 있다. C-interpreter를 통해 C 스크립트로 초기화 과정을 표현할 수 있으며 C 문법의 기본 연산자 및 표준 라이브러리의 사용은 초기화 과정을 간단히 하며 다양하고 유연한 초기화를 가능하게 한다. 이는 기존의 시나리오 파일을 이용한 초기화보다 유연하다. 또한 C 스크립트를 사용해 자동적고 동적인 초기화가 가능하기 때문에 실험계획법에 의한 시뮬레이터 설계에 있어 반복적인 초기화가 요구되는 점을 지원하기에 적합함을 보였다. C-interpreter를 사용해 시뮬레이터의 유연한 초기화가 가능함은 보였지만, 아직 시뮬레이터의 초기화 모듈을 구성하는 C-interpreter 인터페이스가 기본형과 구조체 변수만을 지원하기 때문에 데이터 전송에 부족함이 있으므로 C-interpreter 인터페이스의 추가적인 확장을 필요로 한다.

참고문헌

[1] B.P. Zeigler, H. Praehofer, and T.G. Kim, *Theory of Modeling and Simulation*. 2nd ed. USA: Academic Press 2000.
[2] 최창범, and 김탁곤. "가상 에이전트 기반 시뮬레이션 소프트웨어의 쾌속프로토타이핑 지원 환경."
[3] Park, Gyung-Jin. Analytic methods for design practice. Springer Science & Business Media, 2007.
[4] CINT, https://root.cern.ch/drupal/content/cint accessed : Feb. 2015
[5] Tag Gon Kim, *DEVSimHLA User's Manual*, 2007. Available: http://smslab.kaist.ac.kr accessed : Feb. 2015