[붙임4]

과제 유형	일반 캡스톤디자인

캡스톤디자인(종합설계) 과제 결과 보고서

개설학과	컴퓨터공학과	학기	□1학기 □1·2학기 연계 ■2학기 □하계 □동계
교과목명	캡스톤디자인 II	학점	3
과 제 명	GoLang기반 이산 사건 시뮬레이션 엔진 개발 및 활용		
지도교수	소속(학과)	컴퓨터공학과	성명	최창범

구 분	소속(학과)	학번	학년	성명	연락처	E-mail
참여학생 (총 명)	컴퓨터공학과	20171579	4	김범수	010-4611-0951	skdkwl900@gmail.com
	컴퓨터공학과	20171581	4	도용주	010-7464-7022	qud6330@naver.com
	컴퓨터공학과	20171593	4	이제혁	010-4935-4694	wpqur4694@naver.com

과제요약	모델링 & 시뮬레이션 은 시간, 인력, 자원 등의 낭비없이 다양한 문제 해결에 기여 할 수 있기 때문에 국방, 자율주행, 메타버스 등 다양한 분야에서 사용한다. 하지만 기술이 발전함에 따라 규모와 복잡성의 증대로 결과를 도출 해내기 까지 많은 시간이 소요된다. 이러한 문제를 해결하기 위해 속도가 빠른 GoLang과 경량쓰레드인 GoRoutine을 활용해 이산사건시스템 명세 형식론 에 기반한 시뮬레이션 엔진을 개발하고 사례개발을 통해 엔진의 유효성을 검증하고 활용방안을 제시한다.

위와 같이 캡스톤디자인(종합설계) 과제를 수행하였으며, 결과보고서를 제출합니다.

2022. 12. 05.

지도교수 : 최창범 (인)

대표학생 : 김범수 (인)

한밭대학교 산학협력교육원장 귀하

캡스톤디자인(종합설계) 과제 수행 보고서

1. 과제(주제) 선정 배경 및 필요성

- **모델링 & 시뮬레이션 공학이란**

- 자연현상, 사회과학, 공학 분야의 문제를 모델로 만들고 시뮬레이션 하는 과정이다. 자동차, 드론 등 시스템의 행동을 모델링 해서 컴퓨터 시뮬레이션 엔진을 통해 재현하고 그 결과를 도출해 내어 시뮬레이션 상황을 분석하고 실제 상황에 활용할 수 있다.

- **이산사건시스템 명세(DEVS) 형식론**

- 이산 시간 사건 모델링을 위한 수학적 틀로 DEVS는 최소단위인 원자 모델과 결합 모델로 이루어 져 있으며 원자 모델은 시스템의 행동을 기록하고 모델의 상태에 따른 동작에 대해 표현한다. 결합 모델 은 모델들을 내부적으로 연결하여 만든 모델로 결합 모델을 통해 복잡한 시스템의 모델링이 가능하다.

- DEVS 형식론은 검증된 방법론으로써 여러 분야에서 DEVS 형식론 기반으로 솔루션을 개발하였다. 기존 DEVS 형식론으로 모델링된 모델들을 이식하여 활용 할수 있을 것으로 보인다.

- **시뮬레이션의 필요성**

- 사람의 안전과 밀접한 연관을 지니는 자동화 시스템은 다양한 형태의 확인과 검증이 필요하다. 최근 떠오르고 있는 자율주행 또한 시스템이 주변 환경을 올바르게 인식하고, 모든 행위자의 안전을 보장하는 데 필요한 조치를 취할 수 있는지 엄격한 테스트를 거쳐야 한다. 하지만 아직 검증이 안된 자율주행차를 운행하는 것이기 때문에 실제 도로에서 검증은 진행하는 과정에서 위험한 상황이 발생할 수 있다. 이 같은 문제를 해결하기 위해서 시뮬레이션은 최소한의 리소스로 수만 가지의 상황에 대한 시나리오를 검증할 수 있다.

- **과제 목표**

- 사람들은 규모가 크고 복잡한 시뮬레이션 환경에서도 빠르게 결과를 도출할 수 있는 방안에 대해 연구를 진행하였고 해결 방안으로 프로그래밍 언어적 차원에서의 성능 향상과 시뮬레이션 알고리즘 개선을 통한 성능 향상 이 제시되었다. 본 캡스톤 디자인의 목표는 이 두 가지 방법에 모두 접근해 빠른 성능과 확장성을 보장해 주는 이산 사건 시뮬레이션 엔진을 개발하고 사례개발을 통해 엔진을 검증 및 활용방안을 제시하는 것을 목표로 한다.

2. 과제(주제) 설계 내용 및 추진 방법

■ 개발 방향

- DEVS형식론의 시뮬레이션 알고리즘을 구현
- 컴파일러 언어인 Golang을 이용해 프로그래밍 언어차원에서의 성능향상과 경량쓰레드인 Go루틴을 활용한 알고리즘 차원에서의 성능향상
- 적절한 사례개발을 통해 엔진의 활용방안 제시 및 엔진 검증

■ 이산사건 시뮬레이션 엔진구현을 위한 시뮬레이션 엔진 설계

• 시뮬레이션 엔진 설계 및 아키텍처

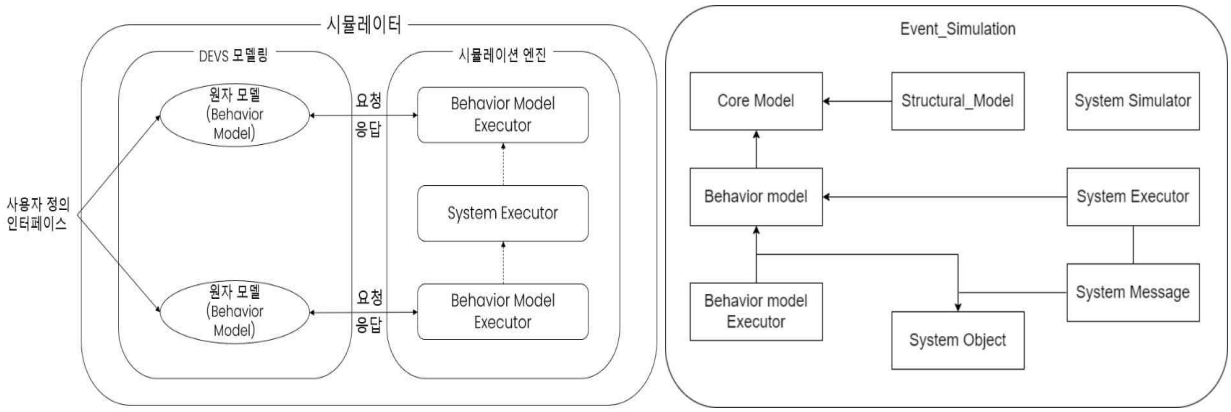
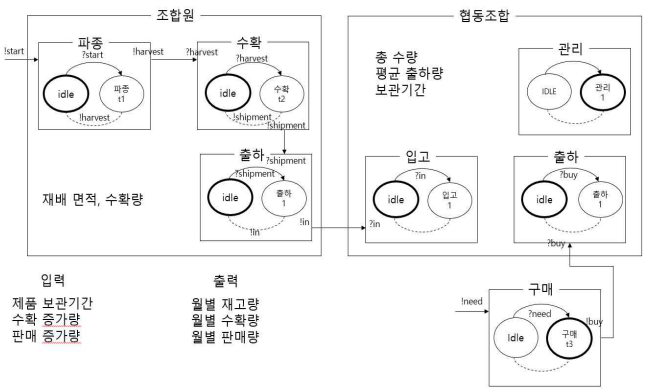


그림1. 엔진 아키텍처

- 모델에 외부로부터 입력이 들어오면 엔진의 Behavior Model Executor 가 원자 모델의 외부 천이 함수를 동작시킨다. 그리고 현재 상태의 임계 시간에 도달하면 내부 천이 함수가 호출되고 이어서 출력 함수가 동작하면 Behavior Model Executor 가 이벤트를 받아 중계 역할을 하는 System Executor를 통해 목적지에 해당하는 모델의 Behavior Model Executor 에게 값을 넘기고 값을 받은 Behavior Model Executor 가 해당 모델에 이벤트를 입력하고 외부 천이 함수를 실행시킨다.
- 성능 개선을 위해 엔진을 분석하는 과정에서 모델의 수가 많아짐에 따라 모델을 정렬하는 부분에서 시간이 가장 오래 걸린다는 사실을 알고 이를 해결하기 위해 스케줄링 된 시간에 따라 모델을 오름차순으로 가지고 있는 자료구조를 기존 dequeue에서 Golang의 동적배열인 Slice로 바꾸고, Golang의 경량스레드인 Go루틴을 활용해 동시/병렬 적으로 정렬을 수행하는 패키지를 이용해 정렬하도록 했다.

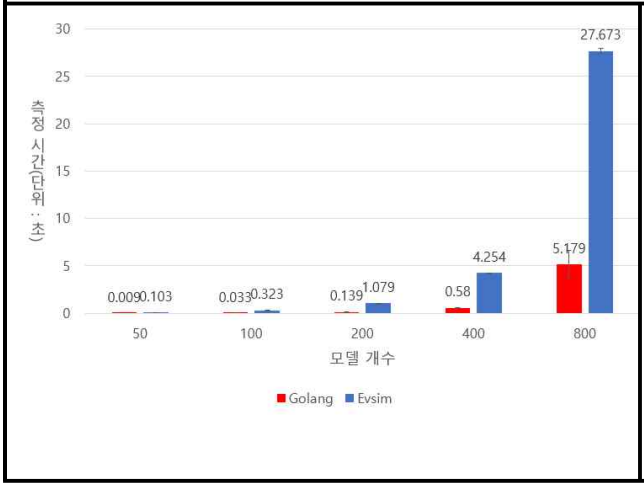
• 사례 개발



- 협동조합을 위한 재고관리 프로그램은 재고파악과 자산관리가 어려운 개발도상국 영세농가를위한 프로그램으로 시물레이션을 통해 협동조합의 수확량을 예측해 재고를 파악하고 공급을 조절 할 수 있도록 한다. 이를 통해 협동조합의 이익을 최대화하고 협동조합이 경쟁력을 가질 수 있도록 돕는다.

3. 과제(주제) 활동 결과 및 결과물(시제품)

본 과제의 결과물로 완성한 시뮬레이션 엔진은 이산사건 시스템 명세 형식론 의 시뮬레이션 알고리즘을 구현하였으며 성능개량을 통해 기존 엔진 대비 빠른 성능을 제공한다.
사례 개발로써 개발한 재고량 시뮬레이션은 조합원, 협동조합, 구매자의 행동과 상태를 이산사건 시스템 명세 형식론 기반으로 모델링하였으며 협동조합의 의사결정에 도움을 주기 위한 시뮬레이션으로써 제작하였다.
제품의 유통기한, 제품의 수요 증가량, 제품의 공급 증가량을 입력으로써 넣게 되면 협동조합의 1년간 입고량과 출하량을 바탕으로 다음 1년을 시뮬레이션 해보고 그 결과로 입고량, 출하량, 총 재고량, 보관기간이 지나 버려진양, 구매를 희망했으나 재고가없어 버려지는양을 시각화하여 보여준다.



5. 결과물의 활용방안(사업화 연계방안)

본 과제의 결과물인 GoLang기반의 시뮬레이션 엔진은 사용자가 자신의 의도에 맞게 모델을 정의하고, 구현하여 시뮬레이션을 수행하게끔 하는데, 많은 수의 모델을 생성해도 빠른 성능을 제공해 사용자가 빠른 시뮬레이션의 결과를 도출하도록 돕는다.
엔진을 활용해 본 과제에서 사례개발로 개발한 협동조합을 위한 재고관리 시뮬레이션처럼 시스템의 행동과 상태 등을 이산사건시스템 명세 형식론 기반으로 모델링하고 시뮬레이션하여 의사결정과 정책결정 을 하는데있어 활용하거나, 거대한 프로젝트에도 활용되서 사용자에게 뛰어난 성능을 제공 할 수 있고 이미 만들어져 국방 등 다양한 분야에서 활용되고 있는 이산사건시스템 명세 형식론 기반 모델을 그대로 이식해서 사용할 수 도 있다.

[붙임5]

캡스톤 디자인 II 최종결과 보고서

프로젝트 제목(국문): Golang 기반 시뮬레이션 엔진 개발 및 활용
프로젝트 제목(영문): Developing and utilizing of Golang-based simulation engines.

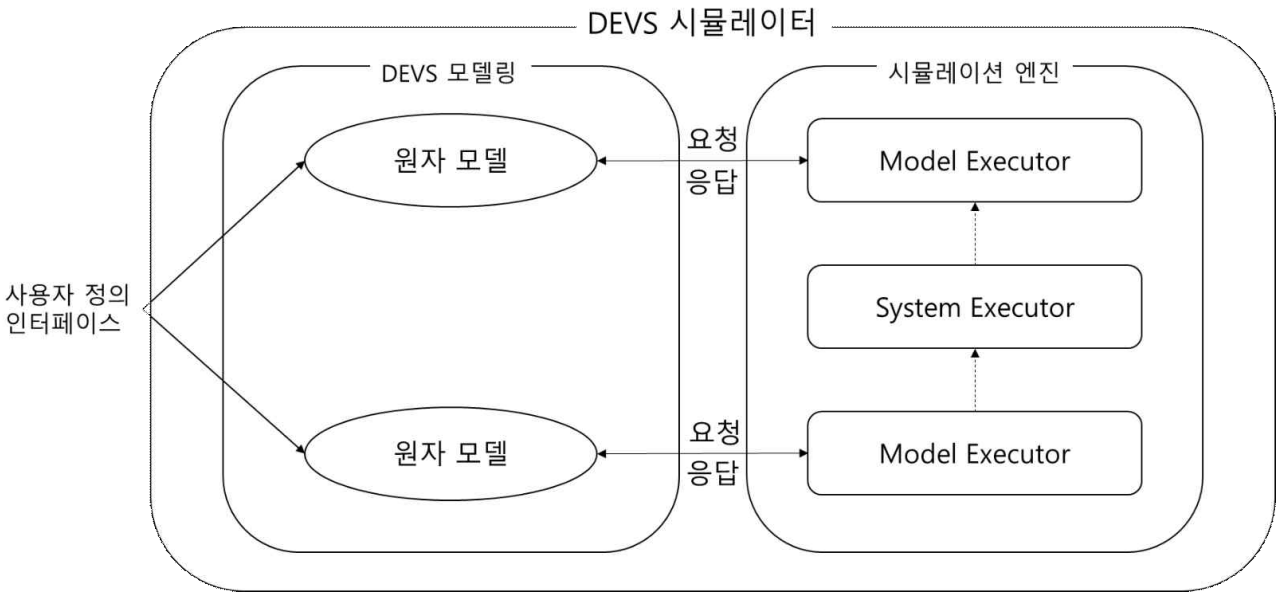
프로젝트 팀(원): 학번: 20171579 이름: 김범수
프로젝트 팀(원): 학번: 20171581 이름: 도용주
프로젝트 팀(원): 학번: 20171593 이름: 이제혁

지도교수: 최창범

1. 중간보고서의 검토결과 심사위원의 '수정 및 개선 의견'과 그러한 검토의견을 반영하여 개선한 부분을 명시하시오.

없음

2. 기능, 성능 및 품질 요구사항을 충족하기 위해 본 개발 프로젝트에서 적용한 주요 알고리즘, 설계방법 등을 기술하시오.



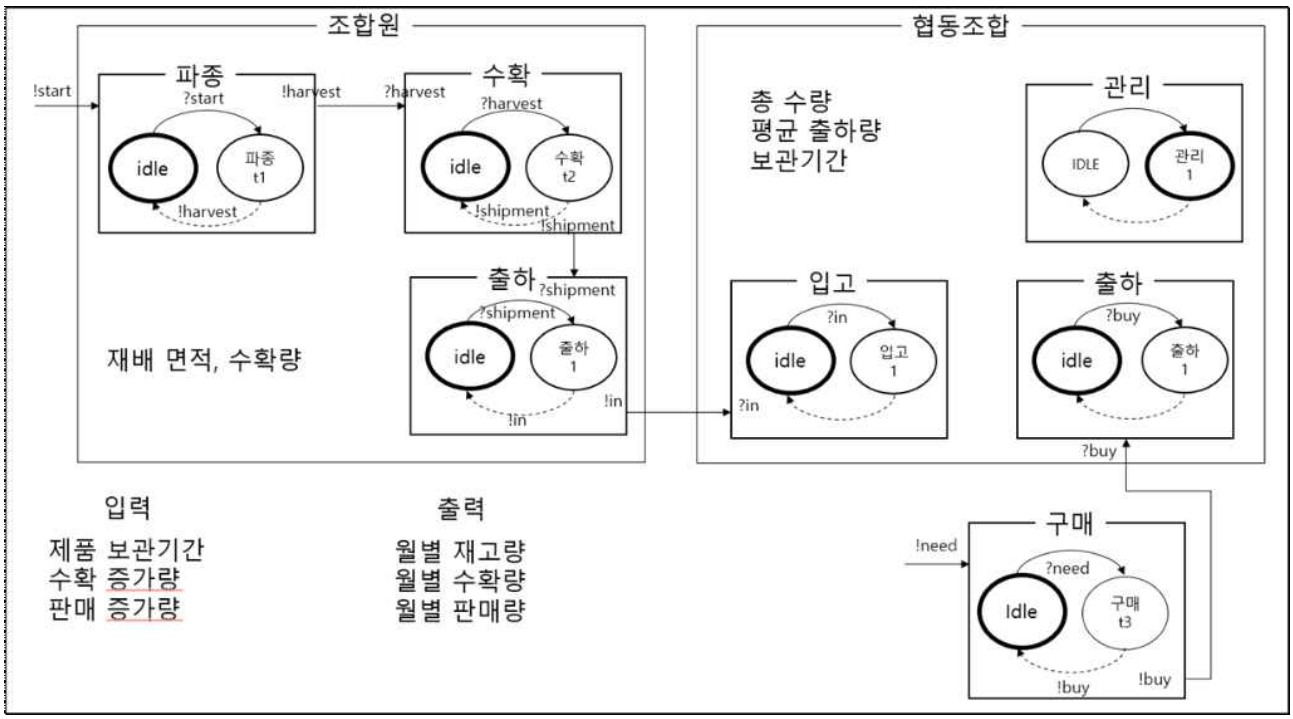
본 시뮬레이션 엔진은 이산사건시스템명세 형식론 기반의 이산사건 시뮬레이션 엔진으로
엔진의 사용자는 모델을 정의하고 있는 구조체를 임베딩 하고 외부이벤트에 따른 모델의 동작 및 상태변화를 명세하는 외부 천이 함수(Extrnal Transition Function), 일정시간 상태의 변화가 없을 때 동작해 내부의 상태를 바꾸는 내부천이함수(Internal Transition Function), 내부 천이 함수 호출시 호출되어 이벤트를 발생시키는 출력 함수(Output Function)를 정의하는 인터페이스를 구현 함으로 써 원자모델을 생성 할 수 있다. 추가로 사용자는 모델간의 연결, 내외부 이벤트, 모델의 상태를 정의하고 모델을 엔진에 등록하고 엔진을 동작시킨다. 모델에 외부로부터 입력이 들어오면 엔진의 Behavior Model Executor 가 원자모델을 외부천이함수를 동작시킨다. 그리고 원자모델의 출력함수가 동작하면 Behavior Model Executor 가 이벤트를 받아 중계역할을 하는 system Executor를 통해 목적지에 해당하는 모델의 Behavior Model Executor 에게 값을 넘기고 값을 받은 Behavior Model Executor 가 해당 모델에 이벤트를 입력하고 외부천이함수를 실행시킨다.
또한 엔진에서 중요한 부분은 타임스케줄링 이다. 모델을 타임스케줄링 하고 정렬하여 요청시간에 맞게 해당하는 모델들을 실행시켜야한다.

객체지향적 모델링 방식을 Golang에서 구현하는 과정에서 Golang의 특징으로 인해 발생한 문제가 있었다.
1. Golang은 클래스 와 상속을 지원하지 않는다.
2. 추상메서드를 지원하지 않는다.

1. 상속의 경우 구조체 임베딩을 통해 상속과 같은 기능을 하도록 했으며,
2. 추상메서드의 경우 인터페이스를 정의하고 인터페이스를 포함하는 구조체를 모델을 구현할 때 임베딩하여 인터페이스를 구현 하도록 했다.

성능 개선을 위해 엔진을 분석하는 과정에서 모델의 수가 많아짐에 따라 모델을 정렬하는 부분에서 시간이 가장 오래 걸린다는 사실을 알고 이를 해결하기 위해 스케줄링 된 시간에 따라 모델을 오름차순으로 가지고 있는 자료구조를 기존 dequeue에서 Golang의 동적배열인 Slice로 바꾸고, Golang의 경량스레드인 Go루틴을 활용해 동시/병렬 적으로 정렬을 수행하는 패키지를 이용해 정렬하도록 했다.

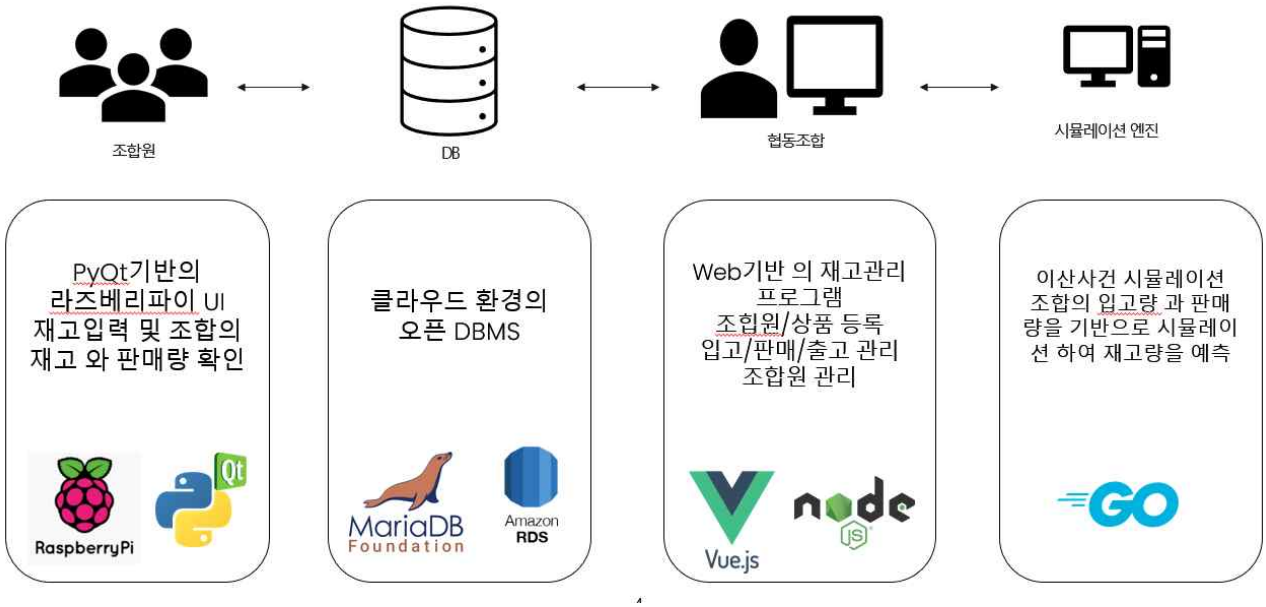
이에 대한 사례개발인 협동조합 재고량 시뮬레이션은 시뮬레이션 엔진을 활용하여,
ERP시스템에서 DB에 쌓이는 조합원의 상태, 입고, 판매 데이터를 토대로 조합원, 협동조합, 구매자의 행동과 상태를 모델링하여 시뮬레이션을 구성하였다.



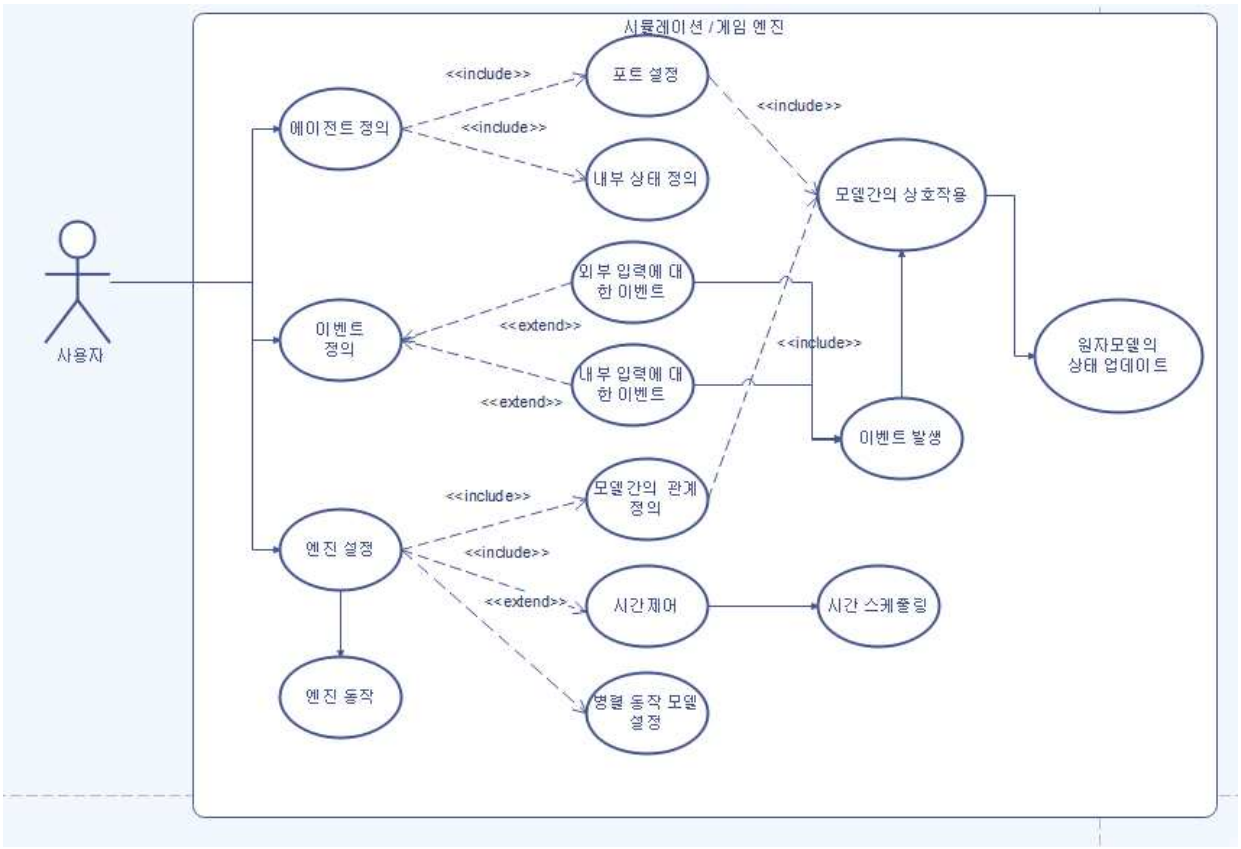
조합원의 재배 면적, 수확량을 조합원의 상태, 입고 데이터를 통하여 설정하게 하였으며, 협동조합은 주기적으로 재고의 보관기간을 관리로 체크하고, 조합원과 구매자의 이벤트를 체크하도록 하였다.
이후 제품의 보관기간, 수확 증가량, 판매 증가량을 입력으로써 받으면, 그에 따른 1년 주기의 시뮬레이션 결과를 보여지게 하였다.

3. 요구사항 정의서에 명세된 기능 및 품질 요구사항에 대하여 최종 완료된 결과를 기술하시오.

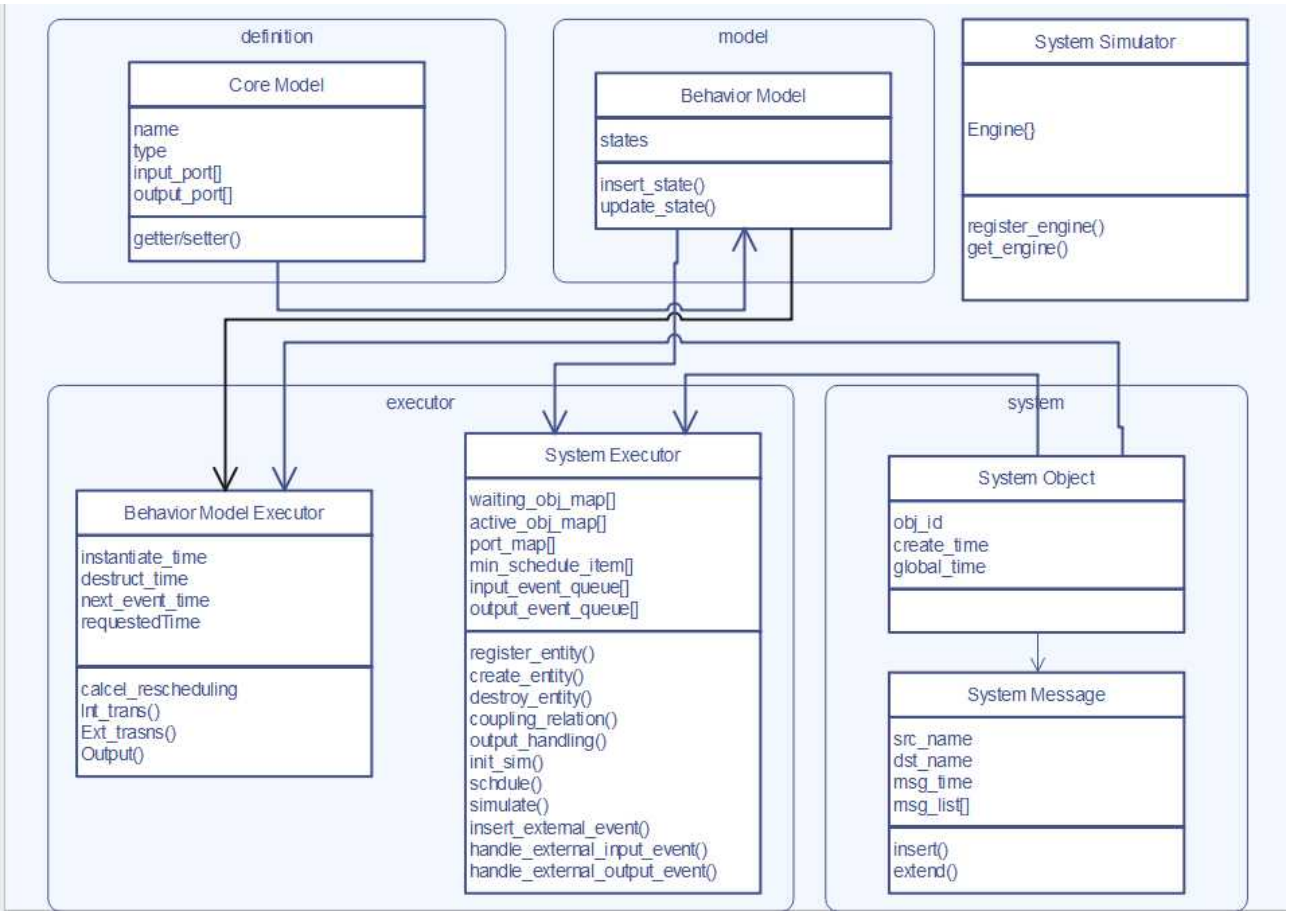
- 소프트웨어(또는 하드웨어, 시스템) 아키텍처 또는 전체 시스템 구성도
  사례개발(협동조합 재고관리 시스템) 시스템 구성도



- 기능별 상세 요구사항(또는 유스케이스)
  시뮬레이션 엔진의 유스케이스

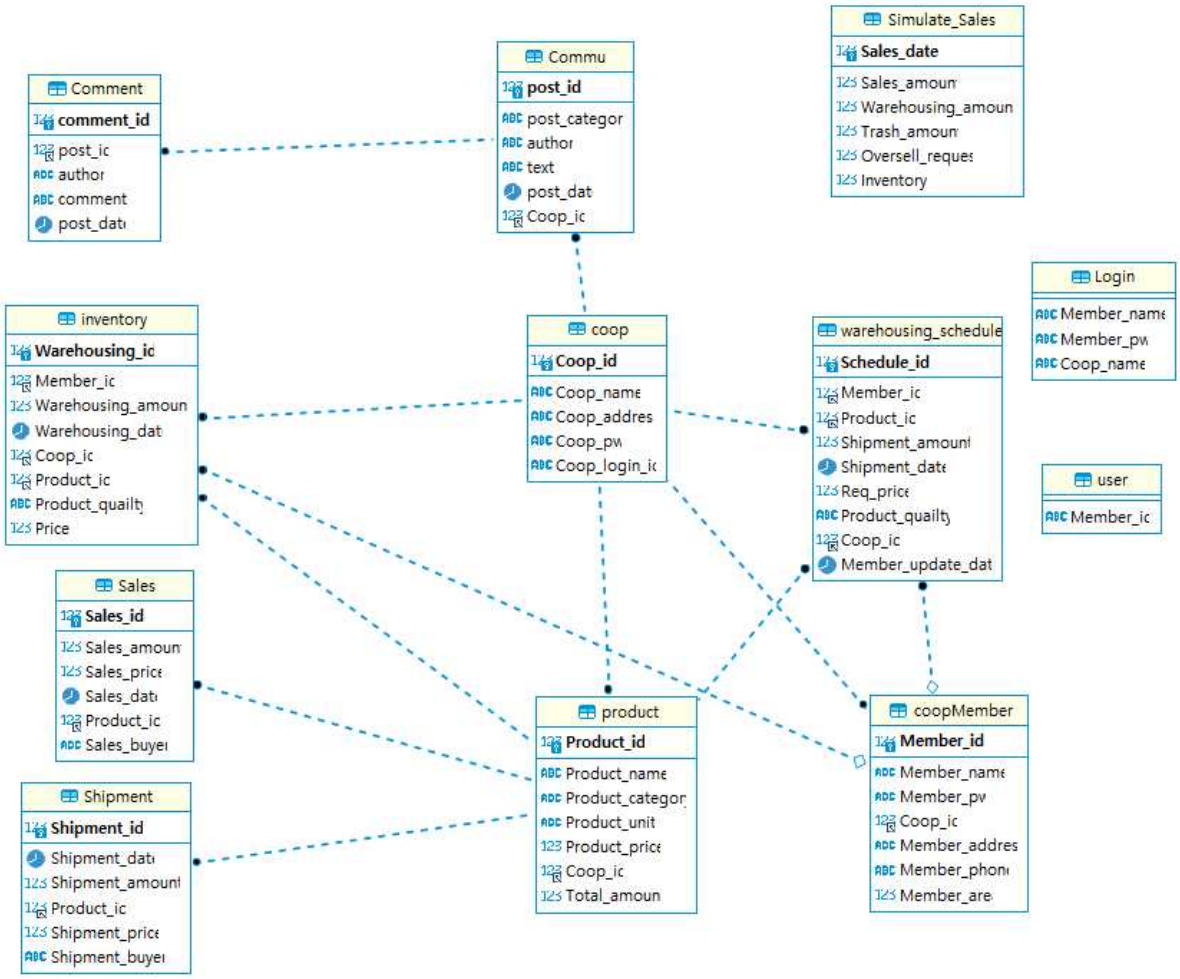


- 설계 모델(클래스 다이어그램, 클래스 및 모듈 명세서 등)
  시뮬레이션 엔진의 클래스 다이어그램



- E-R 다이어그램/DB 설계 모델(테이블 구조)

사례개발(협동조합 재고관리 시스템) DB



4. 구현하지 못한 기능 요구사항이 있다면 그 이유와 해결방안을 기술하시오,

최초 요구사항	구현 여부(미구현, 수정, 삭제 등)	이유(일정부족, 프로젝트 관리미비, 팀원변동, 기술적 문제 등)
		해당사항 없음

5. 요구사항을 충족시키지 못한 성능, 품질 요구사항이 있다면 그 이유와 해결방안을 기술하시오.

분류(성능, 속도 등) 및 최초 요구사항	충족 여부(현재 측정결과 제시)	이유(일정부족, 프로젝트 관리미비, 팀원변동, 기술적 문제 등)
		해당사항 없음

6. 최종 완성된 프로젝트 결과물(소프트웨어, 하드웨어 등)을 설치하여 사용하기 위한 사용자 매뉴얼을 작성하시오.

본 시뮬레이션 엔진은 모델 기반의 이산 사건 시뮬레이션 엔진으로
사용자는 모델을 정의하고 있는 구조체를 임베딩하고
외부이벤트/입력 에 따른 동작함수(Extrnal Transition Function),

외부입력이 없을 때 동작하는 내부동작함수(Internal Transition Function),
내부 동작 함수 호출시 동작하는 출력함수(Output Function)를
정의하여 인터페이스를 구현 함으로써 모델을 생성 할 수 있다.

main.go 작성요령

- 패키지 및 모듈 가져오기

```
package main

import (
        "evsim_golang/definition"
        "evsim_golang/executor"
        "evsim_golang/system"
        "fmt"
)
```

- 모델 정의

1. 사용할 모델 정의

```
type Agent_name struct {
        executor *executor.BehaviorModelExecutor // 모델 구조체 임베딩
        // 필요한 구조 정의
}
```

예시)

```
type Generator struct {
        executor *executor.BehaviorModelExecutor
        msg_list []interface{}
}
```

2. 모델의 생성자 함수 정의

```
func NewAgnet(instance_time, destruct_time float64, name, engine_name string) *Agent_name {
        agn := Agent_name{} // 모델 초기화
        agn.executor = executor.NewExecutor(instance_time, destruct_time, name, engine_name)
        agn.executor.AbstractModel = &agn // 모델 인터페이스 대입
        (외부전이함수, 내부전이함수, 출력함수)
        agn.executor.Behaviormodel.Insert_state(STATE1, time_step) // 상태
        agn.executor.Behaviormodel.Insert_state(STATE2, tiem_step)
        agn.executor.Init_state("STATE1") // 상태 초기화
        // 필요한 생성자 정의
        agn.executor.Behaviormodel.CoreModel.Insert_input_port(agn_input_port) // 입력 포트
        agn.executor.Behaviormodel.CoreModel.Insert_output_port(agn_output_port) // 출력 포트
        return &agn
}
```

예시)

```
func NewGenerator(instance_time, destruct_time float64, name, engine_name string) *Generator {
	gen := Generator{}
	gen.executor = executor.NewExecutor(instance_time, destruct_time, name, engine_name)
	gen.executor.AbstractModel = &gen
	gen.executor.Init_state("IDLE")
	gen.executor.Behaviormodel.Insert_state("IDLE", definition.Infinite)
	gen.executor.Behaviormodel.Insert_state("MOVE", 1)
	gen.executor.Behaviormodel.CoreModel.Insert_input_port("start")
	gen.executor.Behaviormodel.CoreModel.Insert_output_port("process")
	for i := 0; i < 10; i++ {
		gen.msg_list = append(gen.msg_list, i)
	}
	return &gen
}
```

3. 외부이벤트에 따른 모델의 동작 및 상태 천이 함수 정의(Extrnal Transition Function)

```
func (a *Agent_name) Ext_trans(port string, msg *system.SysMessage) {
	// 외부이벤트에 따른 모델의 동작
}
```

예시)

```
func (g *Generator) Ext_trans(port string, msg *system.SysMessage) {
	if port == "start" {
		g.executor.Cur_state = "MOVE"
	}
}
```

4. 내부 동작 함수 호출시 동작하는 출력 함수 정의

```
func (a *Agent_name) Output() *system.SysMessage {
	// 출력 함수 정의
}
```

예시)

```
func (g *Generator) Output() *system.SysMessage {
	fmt.Println(g.msg_list...)
	g.msg_list = remove(g.msg_list, 0)
	return nil
}
```

5. 외부입력이 없을 때 동작하는 내부동작함수 정의

```
func (a *Agent_name) Int_trans() {
	// 내부동작함수 정의
}
```

예시)

```
func (g *Generator) Int_trans() {
        if g.executor.Cur_state == "MOVE" && len(g.msg_list) == 0 {
                g.executor.Cur_state = "IDLE"
        } else {
                g.executor.Cur_state = "MOVE"
        }
}
```

- 메인 정의

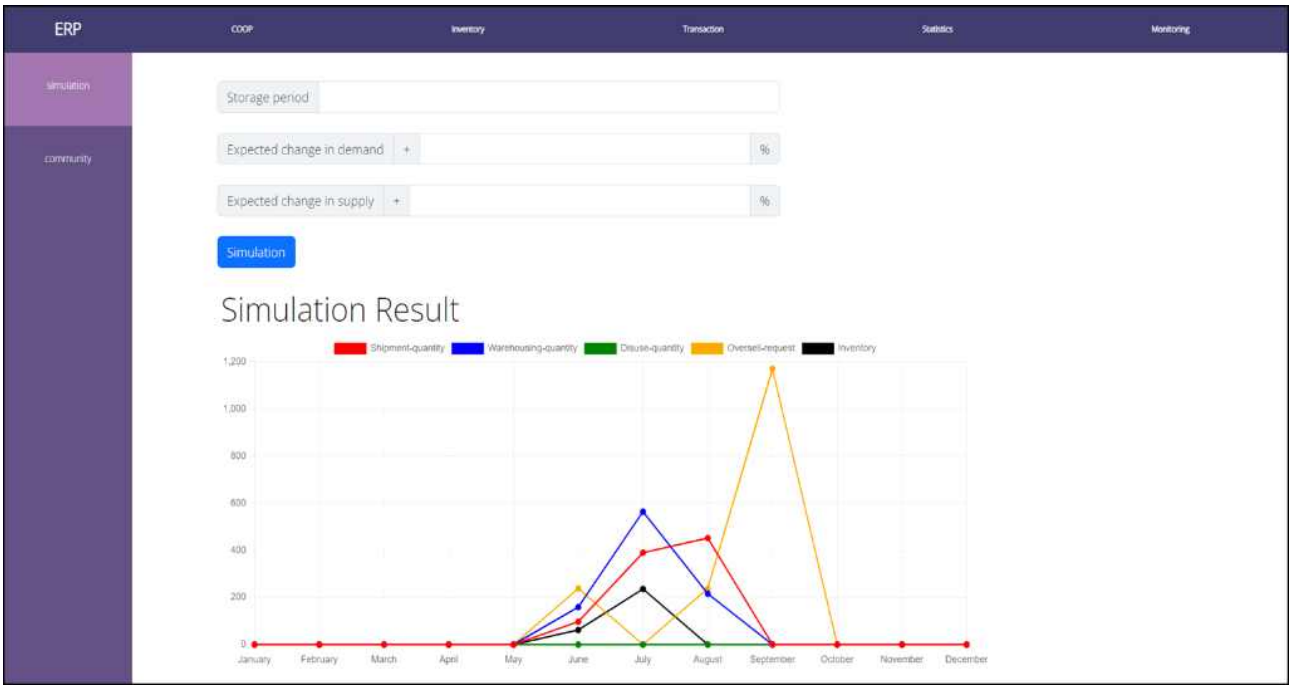
```
func main() {
        se := executor.NewSysSimulator()                    // 엔진 생성
        se.Register_engine(engine_name, sim_mode, time_step) // 엔진 등록
        sim := se.Get_engine(engine_name)                    // 엔진 가져오기

        sim.Behaviormodel.CoreModel.Insert_input_port(input_port) // 입력 포트 생성

        agn := NewGenerator(instance_time, destruct_time, name, engine_name) // 모델 가져오기
        // 오브젝트의 생성시간, 파괴시간, 이름, 등록할 엔진이름

        agn.Register_entity(agn.executor) // 모델 등록

        agn.Coupling_relation(nil, input_port, agn.executor, agn_input_port)
        // 엔진 포트와 오브젝트 포트 연결

        agn.Insert_external_event(port, msg, req_time) // 발생시킬 이벤트 정의

        agn.Simulate(definition.Infinite) // 엔진 실행
}
```

예시)

```
func main() {
        se := executor.NewSysSimulator()
        se.Register_engine("sname", "REAL_TIME", 1)
        sim := se.Get_engine("sname")

        sim.Behaviormodel.CoreModel.Insert_input_port("start")

        gen := NewGenerator(0, definition.Infinite, "Gen", "sname")

        sim.Register_entity(gen.executor)

        sim.Coupling_relation(nil, "start", gen.executor, "start")

        sim.Insert_external_event("start", nil, 0)
```

```
        sim.Simulate(definition.Infinite)
}
```

- 사례 개발 시뮬레이션



사례 개발로써 모델링한 재고량 시뮬레이션은 조합원, 협동조합, 구매자의 행동과 상태를 모델링하여 협동조합의 의사결정에 도움을 주기 위한 시뮬레이션으로써 제작하였다.
제품의 유통기한, 제품의 수요 증가량, 제품의 공급 증가량을 입력으로써 넣게 되면 그에 대한 1년 주기의 시뮬레이션 결과를 시각화하여 보여준다.

시뮬레이션의 결과로 입고량, 출하량, 총 재고, 보관기간이 지나 버려진양, 구매자가 구매를 희망했으나 재고가 없어 판매하지못한 양을 확인 할 수 있다.

7. 캡스톤디자인 결과의 활용방안

시뮬레이션은 현실에서 일어나는 현상과 상황을 분석하고 가상현실에 반영해 구동하여 얻어진 데이터를 바탕으로 시간, 인력의 낭비 없이 위험에 대응하거나, 효율적인 운영에 활용되거나, 다양한 문제 해결에 기여할 수 있다. 그렇기 때문에 시뮬레이션 엔진은 자율주행, 드론, 재난대피, 국방, 도로교통체계 개발 등 여러 분야에서 의사결정 과 정책결정을 하는데 활용되고 있다. 그런데 기술이 발전함에 따라 큰 규모의 복잡하고 계산량이 많은 시뮬레이션 그리고 기술이 발전함에 따라 시뮬레이션의 범위도 광대해졌고 계산량도 많아졌다. 본 캡스톤 디자인의 결과로 완성된 프로젝트는 모델 기반의 이산사건 시뮬레이션 엔진으로 사용자가 자신의 의도에 맞게 모델을 정의하고, 모델을 구현하여 시뮬레이션을 수행하게끔 하는데, 많은 수의 모델을 생성해도 빠른 성능을 제공하여 사용자가 빠른 시뮬레이션의 결과를 도출하도록 도와 의사결정과 정책결정을 하는데있어 활용할 수 있다. 또 기존에 이미 만들어진 다양한 DEVS기반의 모델을 그대로 이식해서 사용할수도 있다.

이에 대한 예시로써 제작한 협동조합 시스템에서 시뮬레이션은 조합원, 협동조합, 구매자의 행동과 상태를 모델링하고 시스템에서 활용되는 조합원들의 상태, 입고, 출고, 판매의 데이터를 토대로 시뮬레이션의 입력으로 적용시켜 1년 주기의 시뮬레이션 결과를 시각화하여 협동조합의 의사결정에 도움을 줄 수 있다.