□ 기술해설 □

소프트웨어 공학적 실시간 시스템 명세를 이용한 가상 환경 구축

포항공과대학교 이지영·김혜정·강교철*·김정현

1. 소 개

가상 현실은 새로운 유형의 컴퓨터 인터페이스라고 생각할 수 있다. 즉 인공적으로 만들어진 환경(Synthetic Environment) 속에서 실생활에서 볼수 있는 객체 또는 은유적인 객체(Metaphorical Objects)들과 직접 자연스러운 방법으로 상호 작용함으로써 가상적인 경험을 얻고. 더 나아가 현 키보드/마우스중심의 인터페이스를 확장하고 정의하는데 목적이 있다. 이를 위해 몰입감(Immersion)을 이용한 텔레프레센스(Telepresence) 및 사실감(Realism)의 구현, 가상 현실 세계와 사용자 간의 상호작용(Interaction), 그리고 실시간 성능의 요구성(Real Time Performance) 등에 대한 고려를 해야 하며, 객체 및 현상들의 시뮬레이션 및 가시화를 위한 에니메이션 기법이 사용되고, 객체들의 형태, 기능, 행위를 효율적으로 모델링하는 것이 필요하다.

효율적인 가상 현실 모델링 방법에 대한 연구는 그래픽스 분야에서 오랫동안 연구되어 왔고, 그래픽스 가상 환경 연구에서 중요한 분야들 중의 하나이다. 그래픽스와 Computer-Aided Design(CAD) 등의 분야에서는 명확하게 형태, 공간적 행위와 기능들을 표현하고, 이들을 모두 통합하여 객체로 표현함으로써 "움직이는 세계"를 만드는 모델링 방법들이 연구되어 왔고, 다년간 여러 도구들이 개발되었다[6. 7, 8]. 그러나, 형태 모델링(Geometrical Modeling)과 행위 모델링(Behavioral Modeling)에 대한 연구가 분리되어 있었으며 이들을 포함하는 가상 현실 시스템에 대한 체계적인 모델링 방법(Systematic Framework)이 마련되어 있지 않다. 대부분의 경우 CAD로 모양을 만든후 일반적인 프로그래밍 언어 수준의 언어를 이용한 프로그래밍을 통해 행위를 넣고 있다. 이러한 구조화되지 않은 개발 방법은 가상 현실 시스템의 개발 과정 및 개발물들의 설정(Configuration)이나 모델이 수정될 때 그것에 따른 적응(Adaptation) 과정을 어렵게 만들고 있다.

소프트웨어 공학분야에서는 실시간 시스템의 명세와 시뮬레이션과 가시화를 통한 유효성 검사(Validation)에 대해 연구해 왔다. 실시간 시스템은 주로 컴퓨터로 만들어지는 제어 시스템(Control System)과 센서와 엑츄에이터 등으로 구성된 제어되는 시스템(Controlled System) 등으로 구성된다. 실시간 시스템은 엄격한 시간적 요구사항들을 가지고 있고 이 요구사항들을 만족시키기 위한 Temporal Logic, 시뮬레이션, 스케쥴링, 태스크 할당(Allocation) 등의 여러 접근 방법들이 고안되었다. 그러나, 대부분의 방법들은 행위와 기능에 초점을 두고 실시간 시스템을 명세하며, 외부 환경이나 형태에 대한 고려를 하지 않는다. 하지만, 대부분의 실시간 시스템들은 그것이 제어하는 동적으로 변화하고 예측하기 어려운 환경에 존재하는 객체들과 물리적으로 복잡하게 상호작용한다. 이러한 실시간 시스템의 의미있는 물리적 시뮬레이션을 위해 제어 시스템과 관계된 실재 객체들은 어떤 모양을 가지고 어느 정도의 공간을 차지해야 한다. 사실, 객체들의 여러 다른 모양과 설정(Configuration)에 의해 시뮬레이션하

*종신회원

면 서로 다른 결과를 만들어 낸다. 예를 들어 외형상 전투기는 여객기와는 다른 유체 역학적(Aerodynamic) 특성을 보인다.

따라서, 시스템의 객체들의 형태, 행위, 기능에 대한 모델링을 동시에 체계적으로 할 수 있는 새로운 개념의 VR 모델러 및 가시화 환경이 필요하다. 이를 위하여 우리는 실시간 시스템 명세와 검증에 대한 소프트웨어 공학의 이론과 형태를 고려하기 위한 설계 이론을 결합하여 형태와 행위, 기능을 통합하여 모델링하고 시뮬레이션 및 가시화를 통하여 검증할 수 있는 체계적인 방법론을 제시하고 ASADAL/PROTO라는 도구를 개발하였다.

ASADAL/PROTO는 기존의 ASADAL[2]이 가지고 있는 행위 및 기능 명세 및 시뮬레이션 기능에 형태 명세를 추가하여 형태, 행위, 기능의 명세를 가시화와 함께 시뮬레이션할 수 있게 확장한 방법 및 이를 구현한 도구이다. ASADAL/PROTO는 ASADAL의 행위, 기능의 정형적인 명세 방법인 Data Flow Diagram(DFD), Time-Enriched Statecharts (TES)와 객체 지향적인 형태 명세 언어인 Visual Object Specification(VOS)을 이용하고 있다. 실시간 시스템의 완전한(Complete) 명세를 위해 ASADAL/PROTO는 DFD, TES와 VOS를 합친 새로운 통합 실시간 객체 모델(Unified Real-Time Object Model)을 제시한다.

2. 시스템 모델링

2.1 개념

형태로의 확장을 하지 않은 ASADAL에서 주요 시스템 모델링 철학은 단계적인(Incremental) 개발 방법의 철학에 기반하며, 그에 적합한 명세 방법과 시뮬레이션 도구를 가지고 있다. 우리는 형태 설계에도 이러한 개념을 확장하고자 한다. 단계적인 개발은 프로그램을 단계적으로 개발하고 유효성 검사할 수 있게 하며, Spiral, Evolutionary-prototyping, Staged-delivery[1] 등의 많은 생명 주기 모델의 기반이 되고 있다.

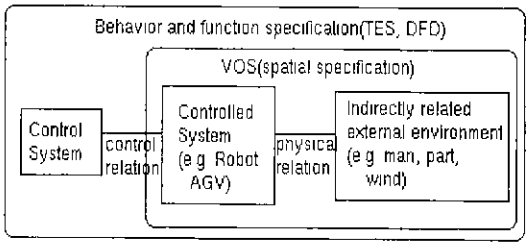


그림 1 제어 시스템(Control System), 물리환경(Physical Principles)과 제어되는 시스템(Controlled System)간의 상호작용으로 나타나는 시스템의 행위

객체들의 형태적 성질들(Physical Properties), 3차원 공간 내에서의 배치(Configuration), 물리적 법칙에 반응하는 행위(Reactive Behavior to Physical Laws) 등 형태에 관련된 속성들은 행위, 기능과 동시에 단계적으로 명세된다. 형태 명세가 제어되는 객체들의 모양과 물리적인 관계들뿐 아니라 환경 객체들도 포함함에 주의해야 한다. 기본적인 생각은 각 차원(즉, 형태, 행위, 기능)의 명세들이 서로서로에게 영향을 준다는 것이다.

우리의 목표는 명세를 여러 상세화 단계에서 시뮬레이션하면서 제어되는 객체와 환경 객체들을 가시화함으로써 개발중인 시스템을 분석, 제어 시스템과 제어되는 시스템, 관련된 환경 객체들간의 상호 작용들을 관찰하는 것이다. 이렇게 하기 위해 제어 시스템 명세의 시뮬레이션은 외부 환경에 대한 시뮬레이션과 동시에 이루어져야 한다.

ASADAL/PROTO의 실시간 시스템에 대한 새로운 모델링 방법은 다음과 같은 세 가지 모델링 차원을 다룬다.

- 행위 명세는 사건(Event)에 대응하는 시스템의 상태 변화를 보여주며, 상태에 따라 적절한 기능들을 활성화(Activation)시킨다.
- 기능 명세는 외부로부터의 입력에 대하여 데이타와 제어 신호의 계산 알고리즘들을 가진다.
- 형태 명세는 객체들의 여러 공간적 성질들을 정의한다. 행위와 기능 명세가 모든 객체에게 주어지는 반면, 형태 명세는 물리적 성질을 갖는 제어되는 시스템(Con-

trolled Device)이나 환경 객체(Environmental Object)에 주어진다.

행위와 기능 모델들은 각각 ASADAL[3]의 명세 방법에서 제공하는 TES(Time-Enriched Statechart)와 DFD(Data Flow Diagram)의 정형적 명세 방법을 사용하여 만들어진다. DFD는 프로세스들을 명세하는데, 이들을 여러 개의 부분 프로세스와 그들의 입력/출력관계로 분할함으로써 시스템의 기능을 밝혀낸다. DFD는 그것을 제어하는 TES를 가질 수 있는데, TES는 어떻게 프로세스가 행동하는지와 시스템의 상태에 따라서 그 프로세스를 언제 실행할지를 명세한다. DFD와 TES를 이용한 정형적 명세 방법의 자세한 설명은 [3]을 참조하라. 형태는 VOS를 사용하여 표현한다. VOS의 주요 목적은 물리적 객체의 형태와 공간상의 정적(Static) 및 동적(Dynamic) 배치를 기술하는 것이다(그림 1 참조).

VOS는 제어되는 시스템이나 실세계 환경 객체들 사이의 물리적 관계나 제약 조건들도 명세한다(예를 들면, Block의 위치는 일단 End-effector에 잡히면 그것의 위치를 따른다).

2.2 모델링 과정과 시뮬레이션에 의한 유효성 검사

ASADAL에서 행위와 기능 명세들은 실제로 몇장의 Message Sequence Diagram(MSD)을 그리는 것으로 시작된다. MSD는 시스템의 객체들 사이에 교환되는 데이타나 제어 신호들을 시간에 따라 나열함으로써 실시간 시스템의

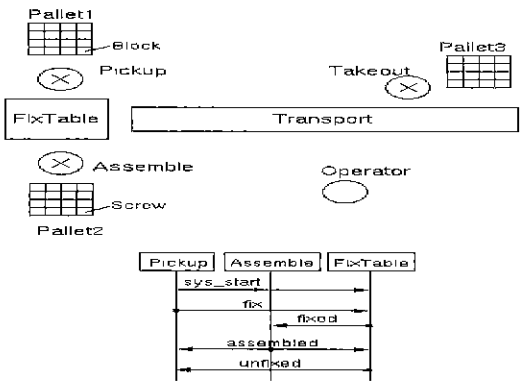


그림 2 한개의 block과 두개의 screw의 조립에 대한 생산 시스템의 설정

외부 행동에 대한 전형적인 시나리오들을 기술한다(그림 2 참조). MSD는 처음부터 객체들을 찾을 수 있게 해 줌으로써 형태에 대한 고려와 함께 전체 모델링 과정에서 객체 지향성을 지원한다. 행위/기능 명세와 형태 설계가 처음에는 각각 서로에게 영향을 주고 받으며 진행되다가 어느 시점에서 검증의 목적으로 시뮬레이션을 효과적으로 수행하기 위해 객체 단위로 결합하므로 이들이 객체들에 대한 일치된 견해를 보이는 것이 중요하다.

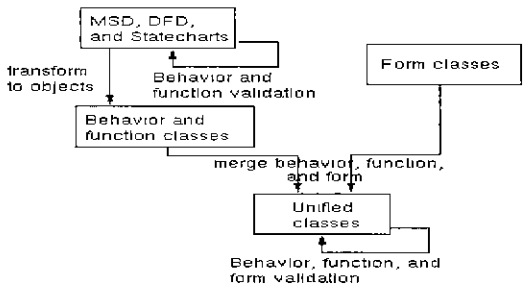


그림 3 단계적인 명세와 유효성 검사 과정

그림 3은 단계적인 방식으로 시스템을 명세하고 유효성 검사를 하는 모델링 과정을 보여준다. 제어 시스템과 실세계 환경을 포함한 전체 시스템의 명세는 그래피컬한 도구인 TES와 DFD를 가지고 만든다. 실세계에 존재하는 객체들에 대한 형태의 명세는 VOS를 사용하여 병렬적으로 수행한다. 객체 지향적 환경을 만들기 위해서 TES를 클래스들의 병렬적인 행위들과 그와 연결된 DFD의 기능들로 나눈다. 이렇게 만든 클래스들 중 로봇같은 것들은 형태를 가질 것이고 제어 소프트웨어와 같은 클래스들은 형태를 갖지 않을 것이다. 그래서 로봇과 같이 행위, 기능, 형태를 모두 가진 클래스들은 행위와 기능 클래스들과 그 형태를 기술한 VOS 클래스들로부터 다중(Multiple) 상속함으로써 만들어진다. 실세계 객체들은 이러한 형태, 행위, 기능 클래스들로부터 설정 인자값(위치, 색깔)들을 새로 할당하고 객체화(Instantiation)함으로써 만들어지고, 형태가 없는 객체들은 행위와 기능 클래스들로부터 객체화된다. 명세가 어떻게 되는지에 대한 상세한 설명은 제3절에 있다.

두개의 시뮬레이션 루프(Loop)가 수행되는

데, 하나는 실시간 제어 시스템의 행위와 기능 명세의 시뮬레이션이고, 다른 하나는 형태를 가진 제어되는 시스템 또는 실제 환경에 존재하는 외부 객체들의 물리적 시뮬레이션이며 가시화와 동시에 이루어지며 상호 작용하는 시스템의 행위가 ASADAL[4]를 사용하여 관찰되고 분석될 수 있다. 이러한 명세와 시뮬레이션에 의한 유효성 검사 과정은 단계적인 방식으로 계속될 수 있다.

3. 명세 방법

이 절에서는 간단한 로봇을 이용한 생산 시스템을 예제로 사용하여 명세 방법에 대해 설명한다. 3.1절에서 중요한 시스템과 환경 객체를 인지하는데 이용되는 방법과 사용되는 요소들에 대해 설명하고, 그 후에 객체에 대한 상세화를 단계적인 방법으로 서로 다른 차원에서 수행하고 나중에 이들을 합치는데 그 과정에 이용되는 명세 방법을 3.2절과 3.3절에서 설명한다.

3.1 시스템 명세

처음에 시스템이 수행해야 하는 "두 부품을 조립하라"라는 기능에 대한 요구 사항으로부터 시작하여, Block을 하나 가져와서 고정시키고 두 개의 Screw를 가져와서 조립한 다음 그것을 저장할 곳으로 수송하고 내려 놓는 세분화된 기능들로 명세한다. 시스템이 그러한 기능을 수행하기 위해서 어떻게 행동해야 할지에 대하여 고려함으로써 그림 2와 같은 초기의 설정을 만든다.

그 과정에서 객체는 컴퓨터에 의해 제어되어 활동적으로 일을 수행하는 로봇과 같은 객체와 활동적이지 않은 Screw와 같은 환경 객체로 구분된다.

실시간 시스템의 외부 행위에 대한 전형적인 시나리오를 기술하기 위해 사용되는 ASADAL에서 소개된 MSD를 시스템의 목적을 완수하기 위해 필요한 시스템 객체와 환경 객체를 더 명확하게 하기 위해 사용한다. 그림 2의 아래에 있는 MSD는 전형적인 시나리오를 가지고 객체 사이의 제어 신호의 흐름을 보여 준다. 예를 들어 그림 2의 MSD는 만약 Block을 FixTable에 옮긴 뒤에 Pickup이 FixTable에 Fix신호를 보내면, FixTable은 Block을 고정한 후에 Fixed신호로 반응한다는 것을 보인다.

시스템을 구성하는 객체들과 그들 사이에 흐르는 데이타 등으로 시스템의 설정을 명확히 한 후 각 객체에 대한 상세한 행위, 기능, 형태 명세를 시작할 수 있다. 그럼 다음 절에서 행위과 기능 명세에 대해 먼저 설명한다.

3.2 행위와 기능 명세

이제 ASADAL의 TES와 DFD를 사용하여 MSD의 각 객체의 기능 구조와 행위를 상세히 명세한다. 먼저, 각 객체의 행위를 따로 명세하지 않고 전체 시스템의 행위를 객체들의 동시 다발적인(Concurrent) 상태들로 명세한다. 그림 4의 TES는 예제 시스템의 행위를 보여준다. 예를 들면, 그림 4의 윗부분에서 보이는 Assemble의 행위는 여러 가지 사건들과 환경 조건들에 의해서 다른 객체들과 동기화되어서 Pallet과 FixTable사이에서 움직이는 상태. Screw를 잡거나 놓는 상태, 조립 작업을 수행하는 상태 등을 가진다. 제어 시스템/제어되는 시스템과 환경 객체들 사이의 관계, 그리고 환경 객체들 사이의 관계는 물리적이므로 간접적인 환경 객체들의 행위는 TES를 사용하여 명세하지 않고, VOS에서 제약 조건으로서 표현될 것이다.

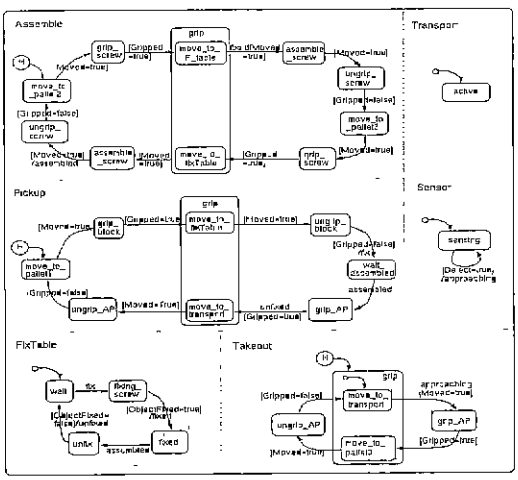


그림 4 생산 시스템 예제의 TES 명세

일단 행위 명세가 TES와 DFD를 사용하여 만들어지면, 명세의 논리적인 일치(Consistency)와 옳음(Correctness), 시간에 맞는 행위에 대한 분석을 할 수 있다. ASADAL의 Subsystem인 ASADAL/SIM은 DFD와 TES 명세들을 실행하고 데이타 흐름에 대한 통계학적 분석, 도달 가능성 분석, 비결정성에 대한 분석 등을 수행한다. 이러한 분석 결과는 형태 명세를 수행하기 전에 행위와 기능 명세가 가진 문제점들을 발견하고 바르게 고치는 데 사용될 수 있다.

나중에 행위와 기능 명세들을 VOS라는 객체 지향적인 형태 명세와 결합하기 위해 TES와 DFD도 그림 5의 왼쪽 윗부분에서 보이는 것처럼 객체 지향적으로 변환해야 한다. 변환된 객체는 TES에 명세된 그것의 행위를 가지고, 어떤 일이 실행되어야 할 때 DFD에 있던 적당한 함수들을 부르며, 상태 전이에 의해 사건이 발생할 때 채널을 통해 신호를 보낸다.

형태가 없는 객체들은 행위와 기능 명세만으로 충분할 것이니 어떤 객체들은 행위와 기능뿐만 아니라 형태를 가진다. 이러한 객체들에 대해 형태는 VOS를 사용하여 명세하며 VOS는 객체들의 물리적인 특성들과 공간적인 행위들을 보인다. VOS에 관한 세부 사항들은 다음 절에서 설명하며 세 가지 차원을 가지는 통합된 객체들을 만드는 방법을 3.4절에서 설명할 것이다.

3.3 VOS

형태 명세는 행위와 기능의 명세와는 독립적으로 진행된다. 예를 들면, 3.1절에서 설명된 시스템 명세로부터 그림 2의 생산 시스템 개요에서 객체들에 대한 대략의 모양/부피와 설정 정보를 잡아냄으로써 형태 명세를 시작한다. 행위나 기능처럼 형태도 VOS를 사용하여 단계적인 방식으로 점차 상세화한다. 다음 절에서 우리는 객체들과 객체들간의 물리적인 성질들, 반응 행위와 공간상의 제약 조건들에 대한 표현 능력 등의 관점에서 VOS에 대한 상세한 설명을 한다.

3.3.1 물리적 성질들과 설정

VOS의 전형적인 속성(Attribute)에는 모양, 재료, 위치, 속도, 가속도, 힘, 각속도, 각가속도, 토크 등이 있다(그림 5 참조). 속성 값들은 사용자로부터 주어지는 프로파일(Profile), 참조 테이블(Lookup Table), 미분 방적식 등 여러 가지 방법으로 표현된다.

VOS 지원 도구에 내장된 라이브러리에는 선가속도, 선속도, 각가속도, 각속도 등과 같은 전형적인 공간적인 움직임을 가진 객체들을 나타내는 메타 클래스들이 있다.

다음의 VOS 클래스들은 두 개의 메타 클래스 SpatialObject와 MovingObject, 그리고 도메인 클래스 AGV(Automated Guided Vehicle)를 보여 준다. 클래스 MovingObject에서 속도는 흐른 시간에 따라 가속도에 맞추어 새 값으로 바뀐다는 것을 보이고 있다(Eq velocity'…로 시작하는 줄 참조). 여기서 작은 따옴표가 붙은 것은 시간이나 계산이 수행된 후의 그 변수의 값을 의미한다. 이러한 시간 관련 방정식은 MovingObject를 상속한 클래스로부터 객체화된 객체의 위치를 시간과 속도에 따라 바꾸어 준다. 예를 들면, 아래에 명세된 AGV에 대해서 새로운 속도는 move 메쏘드에 의해 설정되고, AGV의 위치는 시뮬레이션 동안에 시간이 지남에 따라 바뀌어진다.

```
Class SpatialObject
    Shape shape ;
    Coord location ;
EndClass
Class MovingObject
    Inherit SpatialObject;
    Vect velocity;
    Vect accel;
    Vect angularVel;
    Vect angularAccel;
    …
    Eq location'=location+velocity* time ;
    Eq velocity'=velocity+accel* time ;
    …
EndClass
Class AGV
    Inherit MovingObject ;
    …
```

```
    Method move (Vector newVelocity) :
        velocity'=newVelocity ;
    ...
EndClass
```

객체의 공간적 행위뿐만 아니라 부딛힘, 중력 등과 같이 객체들에 작용하는 공간적 관계와 그들이 따라야 할 물리 법칙들이 있으며 이를 표현하는 방법을 다음 절에서 설명한다.

3.3.2 객체들 사이의 공간적 제약 조건들

실세계 객체들 사이에는 명확한 물리적 관계나 제약 조건들이 존재한다. 예를 들면, Screw는 일단 End-effector에 의해 잡히면 그것의 위치를 따라가게 된다. 이같은 사항들은 VOS로 명세하는데, Screw를 콘베이어 벨트에 붙이는(Attach) 것처럼 객체들이 서로의 공간적인 행위에 영향을 줄 수 있도록 객체들 간의 공간적 제약 조건들로 명세한다. 아래의 예제는 Screw와 같은 움직이는 객체와 콘베이어 벨트와 같은 수송 도구사이에 고려되어야 하는 제약 조건을 보여준다. 그 제약 조건은 일단 움직이는 객체가 수송 도구 표면상에 있으면 그것의 속도는 수송 도구의 표면 속도와 같아진다는 것이다. 여기서 MovingObject는 MovingObject를 조상으로 상속하는 아무 객체를 의미한다. 사실, 완전한 물리적 모델링과 시뮬레이션을 한다면, 예를 들어 표면 저항을 포함한다면, 이런 제한 조건을 줄 필요가 없을 것이나 실제적으로는 모든 가상 현실의 물리적 요소들을 모델링하고 시뮬레이션하는 것은 무리이다.

```
Class Transport
  Inherit FixedObject ;
  Vect surfaceVelocity ;
  Bool active ;
  Method on( ) {active'=True ;}
Method off( ) {active'=False ;}
Constraint On(MovingObject, Transport) and
  active=True→
  MovingObject.velocity'=
      Transport.surfaceVelocity ;
EndClass
```

3.2절에서 설명한 행위와 기능 클래스들과 3.3절에서 설명한 VOS 클래스들은 세 가지 차원을 가진 통합 클래스들로 결합될 수 있다. 다음 절에서 어떻게 그것들이 통합 클래스들로 결합할 수 있는지를 보이고 어떻게 객체들이 시스템 모델을 형성하기 위해 클래스들로부터 객체화되는지 설명한다.

3.4 행위, 기능, 형태를 가지는 통합 클래스

행위/기능 명세와 VOS를 만드는 것이 동시에 진행될 수 있지만, 가시화 시뮬레이션은 제어되는 시스템과 환경 객체들이 세 가지 차원의 명세를 모두 가지기를 요구한다. 그러므로, 제어되는 시스템과 환경 객체들에 대해 그림 5처럼 완전한 실시간 객체 표현을 만들기 위해 적절히 행위/기능 클래스와 VOS를 결합해야 한다. 우리는 이러한 통합 객체들을 행위/기능 클래스와 VOS 클래스를 다중 상속함으로써 만들 수 있다. 그러면 물리적 객체들은 새로운 통합 클래스들을 객체화함으로써 만들어지고, 제어 시스템처럼 형태가 없는 객체들은 행위/기능 클래스들로부터 객체화된다.

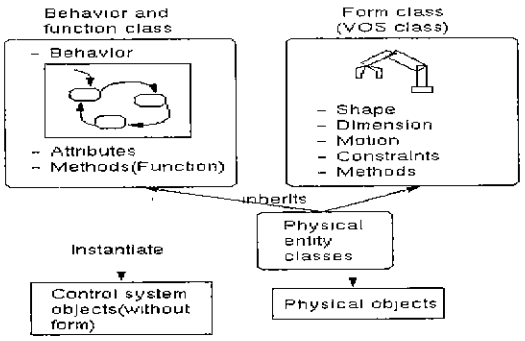


그림 5 행위, 기능, 형태를 명세하는 객체 모델

이런 방식으로 같은 행위를 갖지만 다른 물리적 성질을 갖거나 그 반대 경우의 클래스들을 쉽게 만들 수 있다. 예를 들어 Assemble, Pickup, Takeout 클래스들은 모두 팔과 End-Effector를 가지는 Manipulator VOS 클래스를 상속하지만 그림 5의 행위, 기능 명세에서 만들어진 클래스들로부터 서로 다른 행위와 기능들을 상속한다. Manipulator 클래스는 물리

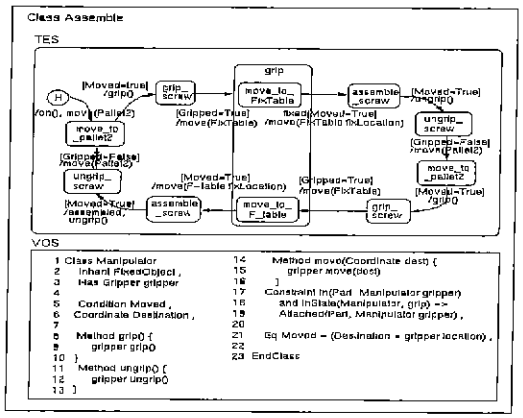


그림 6 물리적 클래스 Assemble

적 세계의 Manipulator의 움직임을 보이는 move 메쏘드를 갖는다. 객체들이 클래스의 객체화에 의해 만들어지며, 필요하다면 여러 개의 Takeout 클래스 객체를 만들 수 있다는 것에 주목하라.

그림 6은 행위, 기능, 형태를 가진 Assemble 클래스를 보여 준다. 이 클래스의 행위와 기능은 그림 4의 TES로부터 변환된 행위, 기능 객체 클래스로부터 상속하고, 그것의 형태 명세는 이미 정의된 Manipulator 클래스로부터 상속한다. Assemble은 VOS에 정의된 move (Pallet2) 메쏘드를 호출함으로써 Pallet2로 움직이고, grip( )을 이용하여 Screw를 잡는다. 움직여진 Screw의 상태는 VOS에 정의된 Moved의 조건 변수로 나타난다. 여기서 In, On, Above 등은 도구에서 제공하는 공간적 관계 조건들이라는 것에 주목하라.

시스템과 그것의 환경에 있는 모든 클래스들이 명세되면 시스템 모델을 만들어야 한다. 객체들은 객체화되면서 초기 설정과 같은 값들이 주어진다.

예를 들면, 그림 2에 있는 물리적 객체들은 위치, 방향 등에 대한 적절한 설정 인자 값들을 가진 Assemble, Transport, FixTable와 같은 통합 클래스로부터 객체화된다. 형태와 환경 객체 행위까지 고려함으로써 전체 시스템 명세의 가시화 시뮬레이션은 훨씬 더 현실적이 되고 개발 초기 단계부터 올바른 시스템 명세의 개발을 돕는다.

4. 결 론

VR 모델은 시스템 개발전에 시스템의 작동 모습을 미리 보면서 유효성을 검증하는 것. 비행 시뮬레이터 같이 시스템과 함께 개발되어 사용자의 훈련을 하는 것 등을 위하여 만들어진다. 따라서 많은 경우 VR 모델은 행위, 기능의 시뮬레이션과 밀접한 관련하에 연동되어야 할 필요성이 있다. 이러한 경우 VR 모델과 행위, 기능 모델은 같이 만들어져야 하며 이들 모두 처음에는 목적과 밀접하고 중요한 것에서부터 상세화해 나가면서 단계적으로 모델링을 해 나가는 방법을 제공해야 한다. "형태는 기능을 따른다"는 말이 있는데, 이것은 시스템의 기하학적, 재료적 성질들이 의미없게 있는 것이 아니고 어떤 효용 가치를 가진다는 것을 말한다[5]. 이러한 형태, 행위, 기능의 유기적인 연결과 이들의 모델링 방법은 그래픽스/VR에서도, 소프트웨어 공학에서도 거의 연구된 바가 없기 때문에 우리는 이 논문에서 이들을 통합하고 시뮬레이션과 가시화를 하는 것이 어떤 식으로 이루어질 수 있으며 어떤 의미를 지니는 것인지를 설명하였다.

ASADAL의 단계적이고 계층적인 모델링과 시뮬레이션 및 가시화 도구들은 객체 지향적인 VR 시스템을 체계적으로 빠르게, 쉽게 만들고 모델의 변화나 시스템 구성의 변화에 쉽게 대처할 수 있게 해준다.

참고문헌

[1] B. W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, pp. 61~72, May 1988.

[2] D. Harel and E. Gery. Executable Object Modeling with Statecharts. In *Proceedings of the 18th International Conference on Software Engineering*, ACM Press, pp. 246~257, Berlin, Germany, March 25~69, 1996.

[3] K. C. Kang and G. I. Ko. PARTS : A Temporal Logic-Based Real-Time Software Specification and Verification

Method. In *Proceedings of the 18th International Conference on Software Engineering '95*, pp. 169~176, Seattle, U.S.A., April 23~30, 1995.

[4] K. C. Kang, M. Christel, J. Herman, J. Morin, P. Place, and D. Wood. Requirements Engineering Project : Project Plan. Unpublished Work, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1991.

[5] W. Mitchell. Reasoning about Form and Function : Computability of Design. Y. Kalay (Editor), John Wiley, 1987.

[6] R. Sriram, R. Logcher, N. Groleau, and J. Cherneff. DICE : An Object-Oriented Programming Environment for Cooperative Engineering Design. *AI in Engineering Design*, Vol. III, pp. 303~366, Academic Press, 1992.

[7] P. Strauss and Rikk Carey. An Object-Oriented 3D Graphics Toolkit. In *Proceedings of the ACM Computer Graphics Conference (SIGGRAPH '92)*, pp. 341~349, 1992.

[8] Java3D API. Sun Microsystems, Inc. Available at 〈http : //java.sun.com/products/java-media/3D/〉.

이 지 영

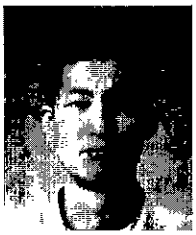
1995 포항공대 전산과(학사)
1997~현재 포항공대 전산과(석사), 포항공대 전산과 박사 과정
관심분야 : Software Engineering, Real-time Simulation 등임

김 혜 정

1997~현재 포항공대 전산과(학사), 포항공대 석사 과정
관심분야 : Virtual Environment Modeling, Software Engineering 등임

강 교 철

1973 고려대학교 통계학과(학사)
1974 콜로라도대학(석사)
1982 미시간대학(공학박사)
1982~1984 미시간대학 교수
1985 Bell Communication연구원
1985~1987 AT&T Bell Labs 연구원
1987~1992 카네기 멜런대 소프트웨어공학연구소(SEI) 수석연구원
1992~현재 포항공대 전산과 부교수
관심분야 : Software Engineering, Formal Specification, Formal Software Analysis 등임



김 정 현

1987 카네기멜런대학 전자 및 컴퓨터공학과(학사)
1989 남가주대학 컴퓨터공학과(석사)
1994 남가주대학 전산과(공학박사)
1994~1996 미국 표준기술연구소(NIST) 연구원 및 NRC Postdoctoral Fellow
1996~현재 포항공대 전산과 조교수
관심분야 : Virtual Environment Modeling, 3D Interaction, CAD 등임