VIP/Sim : Statecharts에 기반을 둔 가상 프로토타이핑 시뮬레이터 설계 및 구현

김 철 웅[†] · 한 상 용[††] · 최 진 영[†††] · 이 정 아[††††]

요 약

시각적인 내장형 시스템 개발 프레임워크(framework)은 가상 프로토타이핑에 기반하여 제시되고 있다. 내장형 시스템은 일상생활에서 종종 신뢰성이 매우 중요한 곳에 사용된다. 시장 적기 진입 시점, 정확성, 사용자 친화적인 디자인 등도 내장형 시스템 디자인에 있어서 고려해야할 필요한 요소들이다. 그러나, 오늘날 내장형 시스템은 비슷한 제품의 이전 경험에 기초한 임시변통적인 접근 방법으로 디자인되고 있다. 이에 새로운 디자인 패러다임이 필요하게 되었고, 이는 정형적인 모델 및 고수준 추상화를 통해 시스템의 행위를 시각적으로 기술할 수 있는 시스템에 기초해야 한다. 가상 프로토타이핑(Virtual Prototyping)은 이런 조건을 만족하면서 정확한 디자인, 명확한 인터페이스 정의, 아이디어 실험, 활발한 의사 소통 등의 장점을 갖는다. 본 논문에서는 이런 가상 프로토타이핑 기술을 실제품 설계, 개발에 이용하기 위한 요소 기술 중 가장 핵심을 이루고 있는 시뮬레이터 일종인 VIP/Sim(VIrtual Prototyping Simulator)의 설계 및 구현에 대해서 기술하고 있다. 특히, 객체지향 방법론의 다형성(polymorphism) 개념을 스테이트 챠트(statecharts)에 도입한 상태 다형성(state polymorphism)을 제안하고, 이를 포함한 확장된 스테이트 챠트를 동적 모델링을 위한 명세 언어로 설계하여 구현에 반영한다. 마지막으로, VIP/Sim을 이용해서 실제 디지털 목업을 개발하는 사례를 보이고 있다.

VIP/Sim : Design and Implementation of Virtual Prototyping Simulator based on Statecharts

Chul-Ung Kim[†] · Sang-Yong Han[††] · Jin-Young Choi[†††] · Jeong-A Lee[††††]

ABSTRACT

A visual development framework for embedded system is presented based on virtual prototyping. Embedded systems often are used in life critical situation, where reliability is very important. Time_to_market, correctness, user_friendly_design are another features required for embedded system design. However, embedded systems are today designed with an ad hoc approach that is heavily based on earlier experience with similar products. We believe that new design paradigm is needed and it should be based on the use of formal model and visual system to describe the behavior of the system at a high level abstraction. Virtual prototyping has all the required features. It has the following advantages; correct design, clear interface definition, idea experimentation, increased communication. In this paper, we describe the design and implementation of VIP/Sim(Virtual Prototyping Simulator), a visionary development framework for embedded system design. New feature such as state polymorphism is augmented to the de_facto standard formal language, statechart, for enhanced dynamic modelling. Actual design experience with VIP/Sim is also discussed.

※ 이 연구는 한국학술진흥재단의 연구지원(과제번호:1998-016-E00060)으로 수행되었음.

† 준 회 원 : 중앙대학교 대학원 컴퓨터공학과
†† 정 회 원 : 중앙대학교 컴퓨터공학과 교수
††† 정 회 원 : 고려대학교 컴퓨터학과 교수
†††† 정 회 원 : 조선대학교 컴퓨터공학부 교수

논문접수 : 1999년 10월 18일, 심사완료 : 2000년 1월 19일

1. 서 론

내장형 시스템을 이용한 응용 제품의 생명 주기(life cycle)는 매우 짧아 신제품 출시 속도는 30% 이상 빨라지고, 복잡도는 매 3년마다 2배정도 증가하고 있으며[1], 전문 설계 인력은 부족해지고 있다. 이러한 어려움들과 함께 시장 적기 진입(time-to-market)이 제품의 성패를 좌우하는 현대의 치열한 경쟁 환경 하에서 기존의 시스템 개발 방법의 근본적인 변화를 요구하는 새로운 패러다임이 필요하게 되었다. 이와 같이, 정보화 사회의 제품 개발에 있어서 시장 적기 진입과 프로슈머(prosumer)를 만족하는 제품 개발은 시장에서의 제품에 대한 성공 여부와 직결된다. 따라서, 제품의 아이디어 창출부터 실제품 출시까지 모든 프로세스가 순차적인 방법으로 이루어지는 기존의 시스템 개발 방법에 근본적인 변화를 요구하는 새로운 패러다임이 필요한데, 이를 충족시키면서 가장 최근에 주목받고 있는 기술이 가상 프로토타이핑(Virtual Prototyping, 이하 VP라 함) 기술이다. 실제품 또는 제품 아이디어와 동일한 물리적 특성(외관, 소리 등), 기능 및 동작을 가지면서, 시뮬레이션이 가능한 디지털 목업(digital mock-up)을 가상 프로토타입(Virtual Prototype)이라고 하며, 이를 구현하여 제품 설계, 개발 및 지원의 전과정을 제어함으로써, 소비자가 가장 원하는 상품을 가장 빠른 시간내에 개발할 수 있는 환경을 구축하는 기술이 가상 프로토타이핑 기술이다[2, 3]. 이런 환경을 구축하기 위해서는 구현된 디지털 목업을 시뮬레이션 할 수는 시뮬레이터, 시뮬레이션으로 검증을 마친후 실제 프로토타입에 필요한 코드를 자동으로 생성하기 위한 코드 자동 생성, 실제품과 동일한 외관으로 현실감 있는 사용자 인터페이스를 지원하기 위한 가시화 기술 등의 요소 기술이 필요하다.

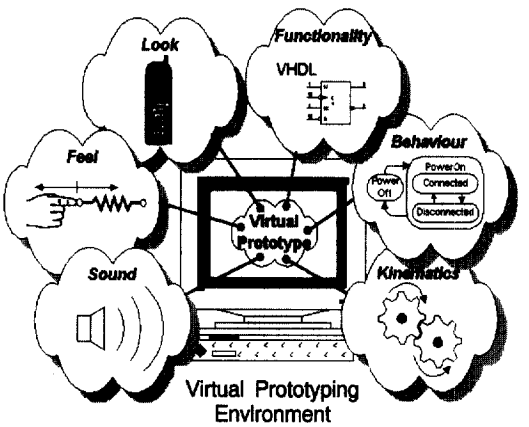
이와 관련하여 국내에서는 체계적이고 안정적인 연구가 이루어지지 않고 있지만, 미국과 유럽에서는 학계를 비롯하여 특히 세계적인 가전 및 통신 회사들을 중심으로 활발히 연구가 진행 중이거나 일부 기능들이 실용화되어 이들 산업에 적용되고 있다. Emultek사의 Rapid+[5], i-logix사의 Statemate MAGNUM[14] 등이 이에 속한다. Rapid+는 내장형 시스템의 가상 프로토타입의 모델링, 시뮬레이션, 코드 자동 생성을 지원하는 개발 툴이다. 가상 프로토타입에 대한 모델링과 시뮬레이션은 스테이트 챠트를 기반으로 하고, 코드 자동 생성은 HMI(Human Machine Interface) 수준에서 C코드를 자동 생성해 주고 있다. Statemate MAGNUM은 복잡한 내장형 시스템 개발을 위한 시각적인 시뮬레이션을 지원하는 툴로써, 이는 시스템의 내재된 기능 및 시스템의 행위를 명확하고 간결하게 하는 시각적인 스펙으로 실행 가능한 모델을 쉽고 빠르게 만들어 준다. 시스템 모델링과 시뮬레이션은 스테이트 챠트를 기반으로 한다. 소프트웨어 개발자를 위해 C나 Ada 코드를, 하드웨어 개발자들을 위해 VHDL 또는 Verilog 코드를 자동 생성해준다. 그러나, Rapid+는 내장형 시스템에 적합한 컴포넌트들이 제한적이고, 사용된 명세언어는 복잡한 시스템의 다양한 행위에 대한 시뮬레이션에는 한계가 있다. 또한, Statemate MAGNUM은 기존의 스테이트 챠트 개념을 확장한 Statemate라는 명세언어를 사용하여 시스템 명세를 여러 관점에서 지원한다는 장점이 있지만, 명세가 복잡해져서 시스템 이해도가 떨어진다. 따라서, 이들이 제공하는 내장형 시스템에 필요한 컴포넌트(component) 라이브러리 한계를 극복해야 하고, 이들이 명세언어로 사용하고 있는 스테이트 챠트(statecharts)를 더욱 개선할 필요가 있다.

본 논문에서는 확장된 스테이트 챠트를 이용한 시뮬레이터를 제안하였고, 이를 설계 및 구현한 것에 대해 기술하고 있다. 개발한 시뮬레이터의 특징은 기존의 툴들이 갖고 있는 내장형 시스템에 필요한 컴포넌트 한계를 극복하여 자유롭게 추가 및 삭제가 가능하게 한 것이다. 또한, 스테이트 챠트에 객체지향 방법론의 다형성(polymorphism) 개념을 도입한 상태 다형성(state polymorphism)을 제안하고, 이를 포함한 확장된 스테이트 챠트를 명세 언어로 설계하고 본 시뮬레이터 구현에 반영함으로써, 시뮬레이션하고자 하는 디지털 목업의 특정 행위에서 존재할 수 있는 다양한 행위들을 쉽고 간단하게 시뮬레이션할 수 있도록 하는 환경을 제공한다. 이로인해, 기존의 툴들보다 더욱 빠르고, 다양하게 제품 설계 및 테스트가 가능하도록 개선되었다.

본 논문의 구성은 다음과 같다. 2장에서는 VP에 관한 전반적인 사항에 대해서 언급한다. 3장에서는 VP를 위한 요소 기술 중 핵심이라고 할 수 있는 시뮬레이터의 일종인 VIP/Sim의 설계 및 구현에 대해서 기술하고 있다. 구현한 VIP/Sim을 이용해서 실제 디지털 목업(digital mock-up)을 개발하는 사례를 4장에서 보이고, 끝으로 5장에서 결론 및 향후 연구 과제에 대해 언급한다.

2. 가상 프로토타이핑

VP 환경으로 만들어지는 디지털 모델(VP 모델)은 (그림 1)에서처럼 외관(look)면에서는 실제품과 동일하고 촉각(feel)과 청각(sound)을 갖추며 기능면에서 제품에 내재하는 전자회로 기능(functionality)과 소프트웨어로 구현될 행위(behaviour)를 포함한다. 또한 기계적 요소(kinematics)도 포함이 되어 실제품이 가질 수 있는 모든 특성들을 디지털화 시킨 모델이다. 이 모델은 VP 기술로 실제 제품에 필요한 코드로 자동 생성될 수 있고, 현실감 있는 가시화 기술로 실제품과의 괴리를 줄일 수 있다.

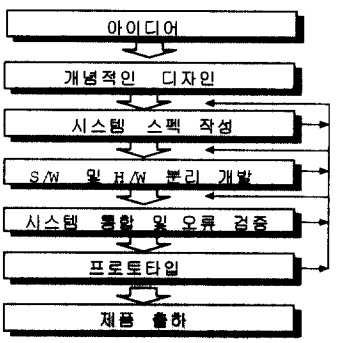


(그림 1) VP 환경[3]

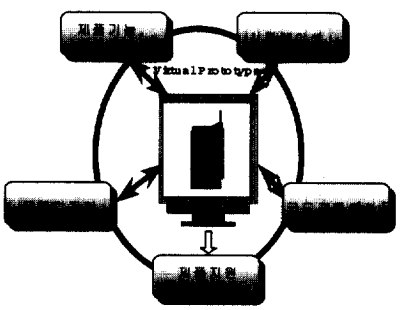
2.1 필요성 및 장점

(그림 2)에서와 같이 기존의 제품 개발 단계는 제품의 아이디어로부터 실제품을 완성하여 출하 전까지의 개발 과정이 순차적으로 하나의 단계가 완료된 후 다음 단계로 넘어가든지, 아니면 미비점 보완을 위해 이전 단계들로 피드백(feedback)된다. 이런 방법의 문제점은 첫째, 실제품을 대량 생산하기전 이에 해당하는 물리적인 프로토타입을 만들어야 하는데, 이에 대한 시간과 비용이 많이 들고, 만들어진 프로토타입의 물리적 특성으로 인해 외형이나 기능을 조금만 변경시키고자 하여도 매우 어려우며, 심한 경우 처음부터 다시 제작에 들어 가야한다. 둘째, 서술식의 모호한 설계 스펙으로 인해 개발 단계에서 자의적으로 해석함으로써 초기 요구 스펙과 상이한 설계를 하여 막대한 비용이 든다. 셋째, 순차적인 개발 단계를 거치기 때문에 전체적인 제품 개발 기간이 길어져 제품의 성능 못지 않게 중요한 시장 진입 시기를 제대로 맞추기 힘들다. 넷째, 초기 단계에 개념적으로 추상화되어 있는 아이디어에 대한 가시화가 쉽지 않아서 이에 대한 다각적인 분석 및 적용이 힘들다. 다섯째, 개발 초기에서부터 제품 출하 시점까지 소비자의 의견을 충분히 수용할 수 있는 수단이 부족하여 사용자 친화적인 상품을 만들기 힘들다.

따라서, 이러한 기존의 순차적인 제품 개발의 단점을 극복하면서 정보화 사회의 제품 개발 방법에 적합한 새로운 패러다임이 필요하게 되었고, 이런 조건들을 만족하는 VP를 이용한 병렬화된 개발 과정의 장점(그림 3)은 다음과 같다.



(그림 2) 기존의 순차적 개발 과정



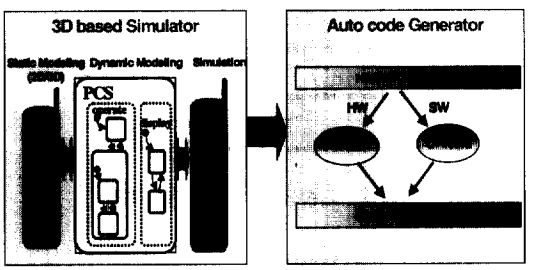
(그림 3) VP를 사용한 병렬화된 개발 과정

첫째, 제품 디자인에 있어서 초기의 아이디어를 빠르게 가시화할 수 있고, 시스템 설계자 중심이 아닌 사용자 중심의 디자인이 가능하다. 둘째, 컴퓨터 기반 시뮬레이션이므로 개발자는 제품을 구성하고 있는 가상의 부품들의 외관이나 기능을 쉽게 변형시키거나 교체함으로써, 다각적인 관점에서 제품의 완성도를 높일

수 있다. 셋째, 제품 개발사이클이 순차적이지 않고 각 개발 단계가 동시에 수행될 수 있으므로 제품 개발 시간 및 비용을 획기적으로 줄일 수 있다. 넷째, VP 기술은 단순히 시뮬레이션하는 수준에만 머무르지 않고 시뮬레이션으로 검증된 디지털 목업은 자동 코드 생성(Code Generation) 등을 통해 전체 시스템에 필요한 소프트웨어와 하드웨어가 통합적으로 구현될 수 있다[5]. 다섯째, 완성된 디지털 목업을 웹상에 올려 놓으므로써, 제품 출하전 소비자에게 선보일 통로를 넓힘과 동시에 호응을 쉽게 알아볼 수 있고, 제품 출하 후 온라인상으로 제공되는 도움 기능으로 제품을 쉽게 사용할 수 있도록 하는 환경을 제공한다[2, 3, 4].

2.2 요소 기술

(그림 4)는 VP 요소 기술인 시뮬레이션, 코드 자동 생성, 가시화(2D, 3D) 등의 기술과 이들간의 관계를 나타낸 것이다.



(그림 4) VP 요소 기술

2.2.1 시뮬레이션

사용자 인터페이스와 모델링 언어를 이용하여 만들어진 시스템 외관과 기능 및 동작을 갖춘 디지털 목업을 시뮬레이션할 수 있는 VP 시뮬레이터가 필요하다. 이는 VP의 핵심적 역할을 수행하는 것으로써, 개발을 위해서 다음과 같은 요소가 필요하다.

첫째, 시스템 설계에 많이 쓰이는 컴포넌트들의 기능을 설계하여 라이브러리화 하는 것이다. 둘째, 구축된 라이브러리를 이용해서 시스템에 필요한 컴포넌트 선정 및 전체적인 시스템 레이아웃을 위한 그래픽 에디터가 필요하다. 셋째, 시스템에 존재하여 시스템을 동작하게 하는 논리를 기술하기 위한 명세 언어가 필요하다. 넷째, 명세 언어에 기술된 내용에 따라 해석 및 실행할 수 있는 시뮬레이션 엔진이 필요하다[7].

2.2.2 코드 자동 생성(Code Generation)

디지털 목업은 시뮬레이션의 검증을 거친 후 분할(Partitioning) 과정을 통해서 시스템 내부에 존재하는 소프트웨어 부분 및 하드웨어 부분으로 나누어진다. 이렇게 분할된 부분은 자동 코드 생성(Code Generation)에 의해 소프트웨어는 C/C++ 등의 코드로 하드웨어는 VHDL 코드로 자동 생성되어 실제 시스템 구현에 필요한 구성요소로 들어가게 된다[5, 8, 9].

2.2.3 가시화

VP 환경에 의해 만들어질 디지털 목업을 실제 사용자와 연결시켜주는 것이 가시화(visualization) 기술이다. 내부적인 시뮬레이션과 코드가 아무리 잘 만들어졌다 하더라도, 사용자에게 현실감 있는 가시화를 제공하지 않으면 VP 기술의 활용도는 그만큼 떨어지게 된다. 따라서, 제품의 전체적인 겉모양 표현과 색상, 표면, 재질들을 표현하기 위한 모델링 및 렌더링 기법, 제품의 역동성을 표현하기 위한 애니메이션 기법, 현실감 있는 사용자 인터페이스 지원을 위한 가상 현실 기법 등 제품의 가시화를 위한 기술이 필요하다[6].

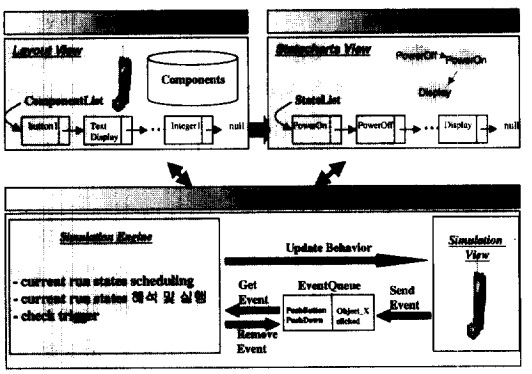
3. VIP/Sim 설계 및 구현

VIP(VIrtual Prototyping)는 VP 환경을 지원하는 통합 도구로써, 이는 크게 VIP/Sim(Simulator)와 VIP/CG(Code Generator)로 구성되어 있다. VIP/Sim은 디지털 목업의 정적 모델링, 동작에 대한 동적 모델링, 그리고 이를 바탕으로 실제로 시뮬레이션할 수 있는 환경을 지원하고, VIP/CG는 시뮬레이션을 통해서 검증된 디지털 목업을 실제 물리적인 프로토타입을 만들기 위해 필요한 코드로 자동 생성할 수 있도록 지원한다. 본 논문에서는 VIP/Sim의 설계 및 구현에 관해서만 기술한다. VIP/Sim은 시스템 개발에 필요한 라이브러리 구축시 컴포넌트의 생성, 제거의 한계를 극복했고, 객체지향 방법론의 여러 개념 중에서 다형성(polymorphism) 개념을 스테이트 챠트(statecharts)에 도입한 상태 다형성(state polymorphism)을 설계 및 구현에 반영함으로써, 다양한 제품 설계 및 이의 다각적인 시뮬레이션이 용이하게 하였다.

3.1 전체 시스템의 구성

본 시뮬레이터의 전체적인 구성은 (그림 5)와 같다.

만들고자 하는 디지털 목업에 대한 컴포넌트 추출, 적절한 배치 등의 정적 모델링을 지원하는 레이아웃 뷰(layout view)가 있다. 그리고, 디지털 목업의 동적인 동작을 기술하기 위해 스테이트 챠트를 이용하여 상태간의 정의 및 상태내에서 해야할 구체적 내용들을 기술할 수 있는 동적 모델링을 위한 스테이트 챠트 뷰(statecharts view)가 있다. 마지막으로, 모델링이 끝난 디지털 목업을 사용자의 반응에 따라 시뮬레이션이 가능하도록 하는 시뮬레이션 엔진과 사용자의 입력 및 그에 따른 시뮬레이션 결과를 보여주는 시뮬레이션 뷰로 구성되어 있다.



(그림 5) VIP/Sim 전체 구성도

3.1.1 정적 모델링을 위한 레이아웃 뷰

```
class CVPTextDisplay {
private :
   CString m_Contents ;
    //Text에 display할 내용을 담고 있는 변수
   COLORREF m_TextColor ;
   // Text color 값
   bool m_ReverseText_flag ;
   // Text 반전 여부를 위한 flag
protected:
   CVPTextDisplay();
public:
   int Contents() ; // 내용을 넣기 위한 함수
   int AddToContents() ; //내용 추가를 위한 함수
   int TextColor() ;    // Text color 값 지정 함수
   void TextBlink() ; // Text 깜빡임을 위한 함수
   int ReverseText() ; // Text 반전을 위한 함수
}
```

(그림 6) Text Display 컴포넌트에 대한 클래스 설계 예

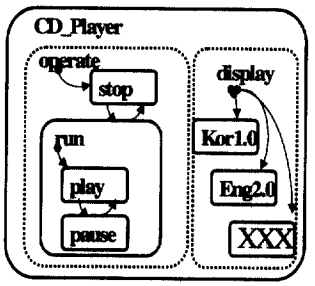
내장형 시스템에 공통으로 존재하는 컴포넌트들을 선정 및 정의하여 미리 구축한 라이브러리(library)에 대해 레이아웃 뷰에서는 만들고자 하는 디지털 목업에 필요한 컴포넌트들을 선택하여 이들에 대한 전체적인 레이아웃 및 각각에 대한 이름, 초기값 등을 설정한다. 이렇게 선택된 컴포넌트들은 리스트화 되어서 추가, 삭제 및 수정이 가능하고, 이런 컴포넌트 리스트는 동적 모델링 또는 시뮬레이션 단계에서 선정된 컴포넌트들에 대한 정보를 제공한다. (그림 6)은 C++을 이용하여 Text Display 컴포넌트를 설계한 예를 보이고 있다.

3.1.2 동적 모델링을 지원하는 스테이트 챠트 뷰

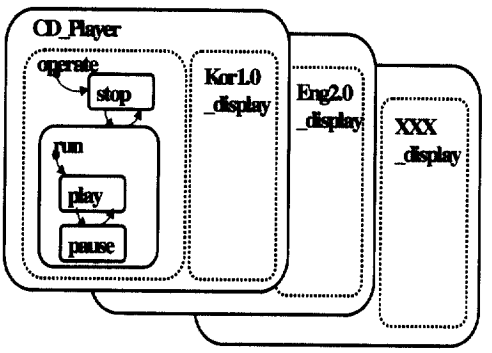
시스템의 동적인 동작을 기술하기 위한 정형 명세 언어가 필요한데, 본 VIP/Sim은 반응 시스템(reactive system)에 de-facto 표준으로 널리 사용되고 있는 스테이트 챠트를 이용했다. 스테이트 챠트는 상태 전이 다이어그램(state transition diagram)에 계층성(hierarchy), 동시성(concurrency), 브로드캐스트 통신(broadcast communication) 등의 개념이 추가되었다. 이를 위해본 시뮬레이터에서 상태(state)들의 관계는 스테이트 트리(state tree) 형태로 표현하여, 상태의 생성, 삭제, 수정, 상태간의 전이(transition) 관계 정의 등이 가능하도록 했다. 특히, 계층성 및 동시성을 표현하기 위해 새롭게 생성되는 상태에서 상태간 계층 관계 및 타입(배타적, 동시적)을 정의할 수 있도록 했다. 또한, 각각의 상태내에서는 트리거(trigger, 이벤트 혹은 조건)에 대한 정의를 할 수 있도록 했고, 트리거가 발생(이벤트 혹은 조건이 만족)하면 전이(transition)가 일어날 목적지 상태(target state)와 이 때 해주어야할 액션(action)을 정의할 수 있도록 했다. 그리고, 현재 상태내에서 실행해야할 행위(activity)를 현재 상태에 처음으로 들어올 때 한번 행해주는 부분, 상태에 머무르는 동안 계속 행하는 부분, 그리고 상태를 빠져나갈 때 해주어야 하는 부분으로 나누어서 정의했다[5, 10, 11, 12].

하지만, 기존의 스테이트 챠트는 각각의 유일한 상태에 대해서 오직 하나만의 구현이 존재하기 때문에, 동적인 동작 모델링 구현에는 어려움이 있다. 즉, 한 스테이트 챠트내에 모두 나타내려면 스테이트 챠트의 복잡도가 증가할 것이고, 여러 스테이트 챠트들에 따로 둔다면 관리의 어려움이 발생한다. 이런 단점들을 극복하기 위해서 객체지향 방법론의 다형성(polymorphism) 개념을 스테이트 챠트에 도입한 상태 다형성

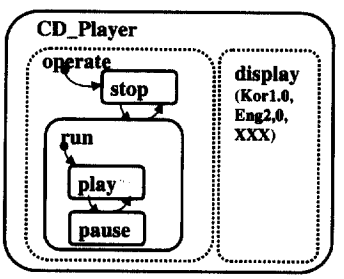
(state polymorphism)을 제안하고, 이를 동적 모델링 설계에 반영했다. 다형성은 동일한 이름을 갖는 연산에 대해서, 각기 서로 다른 구현을 할 수 있도록 한 것으로써, 한 연산에 대해 다수의 클래스가 각각 다른 메소드(method)를 구현할 수도 있고, 한 클래스 내에서 동일한 이름의 연산이 각각 다른 파라메터의 종류와



(a) 복잡도가 증가하는 기존의 스테이트 챠트



(b) 스테이트 챠트들에 대한 관리가 어려워지는 기존의 스테이트 챠트



(c) 제안된 상태 다형성 스테이트 챠트

(그림 7) 기존의 스테이트 챠트와 제안된 상태 다형성 스테이트 챠트

갯수에 따라 정의되고 이용될 수 있다. 이런 다형성의 특징을 스테이트 챠트의 상태에 적용한 것이 상태 다형성(state polymorphism)으로써, 유일한 상태 이름을 갖지만, 여러 가지 구현이 존재하여 동적으로 바인딩(binding)이 이루어질 수 있도록 함으로써, 스테이트 챠트에 대한 복잡도를 단순화시켰고, 또한 동일한 시스템에 대해 여러개의 스테이트 챠트가 필요 없게 했다.

만일, CD_Player의 display 부분이 한국판과 영어판이 필요할 때, (그림 7(a))처럼 하나의 스테이트 챠트 내에 표현한다면 display 상태가 많아져서, 스테이트 챠트 복잡도가 증가할 것이고, (그림 7(b))처럼 각각의 다른 스테이트 챠트로 표현한다면 하나의 시스템에 대해서 여러 스테이트 챠트들이 존재하여 이에 대한 관리가 어렵게 된다. (그림 7(c))처럼 display 상태에 여러 구현이 존재할 경우 파라메터화해서 이를 표현함으로써, 기존의 방식(그림 7(a)), (b)보다 스테이트 챠트를 단순화시킬 수 있고, 명세 및 스테이트 챠트 관리도 훨씬 용이해진다.

이상과 같이 VIP/Sim에서의 동적 모델링을 위한 스테이트 챠트 뷰는 기존의 스테이트 챠트가 갖는 장점을 그대로 살리면서 상태 다형성을 고려한 확장된 스테이트 챠트를 지원해준다. 상태 다형성을 포함한 스테이트 챠트에서 필요한 상태 클래스에 대한 설계는 (그림 8)과 같다.

```
class State {
public :
   CString my_name ;
   // 유일한 상태 이름에서 구현이 다른 상태를 구별 위해
   CString version ;
   int parent_id ;
   int me ;
   int default_child_id ;
   CString child_state_type ;
   //destination state, event, condition, action에 대한 정보
   CDtaList Dtas ;
   // enter에서 해야할 일에 대한 명세
   CEnterList Enters ;
   // loop에서 해야할 일에 대한 명세
   CLoopList Loops ;
   // exit에서 해야할 일에 대한 명세
   CExitList Exits ;
}
```

(그림 8) 상태 클래스 설계

3.1.3 시뮬레이션 엔진과 시뮬레이션 뷰

레이아웃 뷰로부터 만들어진 컴포넌트들에 대한 리스트와 스테이트 챠트 뷰로부터 만들어진 상태들에 대한 리스트 및 각각의 상태들에서 구체적으로 해야할 일들에 대한 내용을 바탕으로 디지털 목업에 대한 시뮬레이션이 가능하도록 하는 것이 시뮬레이션 엔진이다. 특히, 이 시뮬레이션 엔진은 상태 다형성 스테이트 챠트를 고려해서 같은 이름하에서 상이한 버전 이름을 갖는 상태들을 동적으로 연결해가면서 시뮬레이션 할 수 있다. 그리고, 사용자의 반응 이벤트를 시뮬레이션 엔진에게 보내고, 시뮬레이션으로 부터의 동작 지시를 사용자에게 보여주는 것이 시뮬레이션 뷰이다.

시뮬레이션 엔진에 대한 전체적인 알고리즘은 (그림 9)와 같다. 먼저, 스테이트 챠트의 계층성 및 동시성을 고려하여 실행시킬 상태들을 선정한다. 이를 위해 각각 동시성을 가지면서 독립적으로 존재하는 서브 시스템 각각에 대한 상태들을 연결한 리스트(RunStateList)가 있고, 동시성을 가지는 이런 서브 시스템들을 전체적으로 관리하기 위한 배열 구조(RunStateListArray)가 있다. 실행해야할 상태들을 담고 있는 이 배열(RunStateListArray)에 정의된 초기화 부분(enter)을 실행(시뮬레이션 뷰에 반영)하고, 트리거에 의한 목적지 상태(target state)가 결정될 때까지 각 서브 시스템들에 정의된 순환 부분(loop)을 계속 순환하면서 실행한다. 한편, 시뮬레이션 뷰에서 사용자의 행위에 따라 발생한 외부 이벤트(event)는 이벤트 큐(Event Queue)로 보내지고, 시뮬레이션 엔진(simulation engine)은 순환 부분을 지속적으로 순환 실행하면서 동시에 이벤트 큐를 체크하여 현재 실행 중인 상태에서 정의된 이벤트와 같은지를 본다. 또한 실행 도중 타이머와 같은 자신의 내부 이벤트, 혹은 정의된 조건이 만족되고 있는지의 여부도 체크한다. 만일 이벤트 혹은 조건이 만족되면 해당 서브 시스템에 대한 리스트에 정의된 종료 부분(exit)을 실행하고, 목적지 상태에 따른 새로운 실행 상태들에 대한 리스트(RunStateList)를 만들다. 이와 같은 일을 사용자에 의해 시뮬레이션 중지 명령이 일어날 때까지 동일한 방법으로 실행한다. 특히, 실행시킬 상태들을 선정하여 리스트(RunStateList)를 만들 때, 상태 다형성에 해당되는 상태들에 대해 현재 시뮬레이션을 위해 세팅된 버전과 일치하는 상태를 선정하여 실행 리스트를 만듦으로써, 테스트하고자 하는 버전에 맞게 시뮬레이션할 수 있다.

```
Algorithm Simulation_Engine

InitialRunStateScheduling() ;
        //최초로 실행시킬 State들을 스케쥴링한다.
while(!sim_stop_flag) { // simulation 중지가 아닌 동안
   RunStateRescheduling_flag = false ;
        // concurrent한 서브 시스템들에 대해서
   for(sub_system_id = 0 ;
          sub_system_id< sub_system_num ; sub_system_id++) {
        ExecuteLoop(sub_system_id) ;
        // 각 서브 시스템들의 state들에서 반복 실행되는 Loop부분을 실행
        if(target_state = CheckMyTrigger(sub_system_id)) {
        // 서브시스템의 state들에 정의되어 있는 trigger와 일치 여부 체크
           RunStateRescheduling_flag = true ;
           RunStateRescheduling(target_state) ;
           // target state에 맞게 재스케쥴링
           // 이때, 상태 다형성을 고려해서 현재 세팅된 버전과
           // 일치하는 상태를 선정한다.
        }
   }
   if(RunStateRescheduling_flag) { // 재스케쥴링이 되었다면
        ClearExternalEvent() ;
          // 체크한 외부 이벤트를 외부 이벤트 큐에서 삭제
        ExecuteExit() ;
          //exit되는 state들 결정 후 이들의 exit 부분에 기술된 내용 실행
        ExecuteEnter() ;
          // 재스케쥴링으로 새롭게 포함되는 state들을 결정 후
          // 이들의 enter 부분에 기술된 내용 실행
   }
}
```

(그림 9) Simulation Engine 알고리즘

3.2 구 현

구현 환경은 펜티엄 III-450, 128M RAM, MS-Windows NT 4.0, MS-VC++ 6.0 하에서 구현했다. VIP/Sim은 정적 모델링을 위한 레이아웃 뷰, 동적 모델링을 위한 스테이트 챠트 뷰, 그리고, 시뮬레이션을 위한 시뮬레이션 뷰로 구성되어 있다. 레이아웃 뷰에서는 만들고자 하는 컴포넌트를 선정하여 적절히 배치하고, 이 컴포넌트에 대한 이름 및 초기값 등은 다이얼로그 박스를 이용하여 부여할 수 있다. 스테이트 챠트 뷰는 스테이트 트리(state tree)와 스테이트 에디터(state editor)로 구성되어 있다. 스테이트 트리는 상태간의 관계를 명세한 것이다. 상태간의 동시성 여부를 시각적으로 명확히 하기 위해 동시에 실행되는 상태들은 이름 앞에 초록색으로 표시되고, 배타적(exclusive)으로 실행되는 상태들은 노란색으로 표시된다. "새로운 State 만들기", "State 이름 바꾸기", "Destination State 지정하기", "Default State로 만들기", "State 삭제하기", "Global Version Management", "Version Management in a State" 등의 UI가 제공된다. 특히, "Global Version

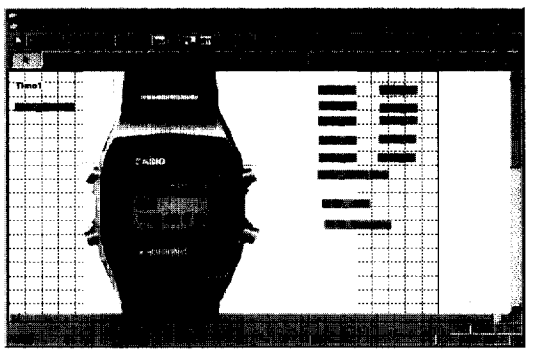
Management" 메뉴는 모델링하고자 하는 디지털 목업의 여러 버전들을 추가 및 삭제할 수 있고, "Version Management in a State"는 정의된 버전 중 현재 선택한 상태와 같은 이름이면서 구현이 다른 형태의 버전을 만들 수 있도록 하는 메뉴이다. 또한, 이 각각에 대한 세부 사항들을 위한 다이얼로그 박스가 있다. 스테이트 에디터는 현재 선택한 상태내에서 구체적으로 해야할 일들을 기술한다. 스테이트 에디터의 상단에는 정적 모델링에서 선택된 컴포넌트들에 대한 정보(이름 및 이 컴포넌트에서 사용 가능한 함수) 등이 나타나고, 이를 이용하여 상태의 논리를 기술하기 위해 필요한 컴포넌트들에 사용될 함수를 클릭하고, 연산이나 치환을 위한 편집을 한 후 해당 영역(event, condition, action, enter, loop, exit)에 넣기만 하면 된다. 마지막으로 시뮬레이션 뷰는 정적 및 동적 모델링으로부터 만들어진 디지털 목업을 실제 시뮬레이션할 수 있도록 하는 부분이다. sim 버튼을 누르면 초기에 정적 모델링에서 배치한 그래픽적인 컴포넌트들이 나타나고, 사용자 반응에 따른 이벤트들을 고려하여 미리 정의된 동작을 (그림 9)의 시뮬레이션 엔진 알고리즘에 의해 실행된다. 특히, 동적 모델링 단계에서 정의한 버전들을 시뮬레이션 도중에 선택하여 여러 버전들에 대해 동적으로 시뮬레이션할 수 있다.

4. VIP/Sim을 이용한 VP 모델(Digital mock-up) 개발 사례

VIP/Sim을 이용해서 소리와 진동 알람 기능이 있는 디지털 시계를 개발 하는 과정에 대한 예를 보이고자 한다. 개발 프로세스는 먼저 디지털 시계에 대한 전체적인 배치를 포함한 정적 모델링을 한 후, 스테이트챠트 명세에 의해 사용자 반응에 따른 동작, 혹은 시계 내부의 동작 등의 동적 모델링을 한다. 이런 모델링이 끝난 후 명세한 대로 실행하는 지의 여부를 검증하기 위해 시뮬레이션한다.

4.1 정적 모델링

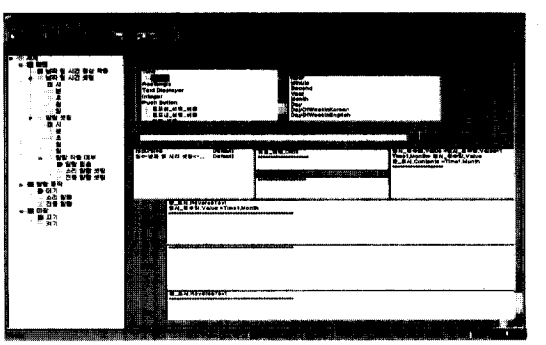
(그림 10)에서와 같이 먼저 VIP/Sim의 레이아웃 뷰(layout view)를 이용해서 디지털 시계에 필요한 컴포넌트(Time, 알람 시간을 저장할 정수들, 현재 시간을 저장할 정수들, Push button, Text Display, Sound, Bitmapping, Grouping 등)들을 선정한다. 이렇게 선정된 컴포넌트들에 대해서 컴포넌트의 이름과 필요하다면 초기값 등을 부여한다. 예를 들면, 비트맵(bitmap) 파일이나 사운드(sound) 파일을 필요로 하는 컴포넌트들은 여기서 파일의 위치 등에 대한 정보를 지정할 수도 있다. 이렇게 선정된 컴포넌트들을 적절히 배치한다.



(그림 10) 디지털 시계의 정적 모델링

4.2 동적 모델링

정적 모델링이 끝난 디지털 시계에 대한 동적 모델링을 위해서 (그림 11)의 왼쪽 부분에 있는 스테이트 트리를 통해서 디지털 시계에 존재하는 상태간의 정보를 명세한다. 특히 만들고자 하는 디지털 시계는 화면 부분, 알람동작 부분, 야광 부분이 서로 무관하게 동작하므로 동시성을 갖도록 명세해야 한다. 오른쪽 부분의 스테이트 에디터를 통해 상태내에서 기술되어야 할 내용들을 명세하는데 계층성을 잘 생각해서 동기(sibling) 상태에서 공통적인 부분은 부모(parent) 상태에서 수행될 수 있도록 한다.



(그림 11) 디지털 시계의 동적 모델링

4.3 시뮬레이션

정적, 동적 모델링이 끝난 디지털 시계는 sim 버튼을 눌러서 시뮬레이션을 시작하고, stop 버튼을 눌러서 시뮬레이션을 종료한다. 디지털 시계에 있는 4개의 푸시 버튼을 눌러가면서 시간 셋팅, 소리와 진동 알람 셋팅 등의 기능이 동적 모델링에서 명세된 대로 제대로 반응하는지를 시뮬레이션한다. 특히, 시계의 디스플레이 부분에서 영어와 한글을 고려하여 만들어 놓은 두 가지 버전을 동적으로 바꿔가며 시뮬레이션한다. 정적 및 동적 모델링으로의 피드백 과정을 반복하여 다각도로 테스트를 시도하면서 시뮬레이션을 진행한 후 최종적으로 원하는 디지털 목업을 만든다.

5. 결론 및 향후 연구 과제

본 논문에서는 내장형 시스템 설계에 있어서 기존의 순차적인 개발 방법의 문제점들을 극복하는 새로운 패러다임인 VP 환경 중 핵심 요소 기술인 시뮬레이터를 설계 및 구현한 VIP/Sim에 대해서 알아보았다.

VIP/Sim은 기존의 다른 툴들을 이용했을 때의 공통된 장점인 제품 개발 기간 단축(5~10배) 효과[13, 14] 및 시각적이며, 통합된 제품 설계로 제품 설계시 발생할 수 있는 오류를 최소화할 수 있었다. 또한, 기존의 Emultek사의 Rapid+[5], i-logix사의 Statemate MAGNUM [14] 등의 툴들을 이용했을 때의 컴포넌트 라이브러리 한계를 극복할 수 있게 되었다. 특히, 객체지향 방법론 중 다형성(polymorphism) 개념을 스테이트 챠트(statecharts)에 도입한 상태 다형성(state polymorphism)을 설계 및 구현에 반영함으로써, 간결한 행위 명세와 다각적인 시뮬레이션의 지원이 가능하여 제품의 다양한 설계 및 테스트가 가능해졌다.

향후 연구 과제로는 첫째, 동적 모델링 명세 언어로 사용된 스테이트 챠트를 시스템 설계가 더욱 쉽고, 명확하도록 하기 위해 더욱 확장하는 것이다. 둘째, 실제 물리적인 프로토타입과 가상으로 만든 프로토타입과의 괴리를 최대한 줄이기 위해 3D를 포함한 가상현실 기법도 수용해야한다. 마지막으로 코드 자동 생성과 같은 VP 기술 요소와도 잘 접목될 수 있도록 연구가 진행되어야 한다.

참 고 문 헌

[1] Hennessy and Patterson, "Computer Architecture 2nd edition," 1997.

[2] P. S. Pulli, M. L. Salmela, J.K. Simila, "Virtual Prototyping based development and marketing of future consumer electronics products," IFIP word computer congress, Sep. 1996.

[3] Mr. Mikko Kerttula and Mr. Marko Salmela, "Virtual prototyping in product development," http://www.ele.vtt.fi/projects/virpi/vpipd.htm.

[4] M. E. Kerttula1, P. J. Pulli1,2, "Product Development, Marketing and Mobility Support in Virtual Enterprise," http://www.ele.vtt.fi/projects/virpi/ice96.htm.

[5] David Harel, HAGI LACHOVER "STATEMATE : A Working Environment for the Development of complex Reactive Systems," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol.16, No.4, APRIL 1990.

[6] 이희웅, 최진영, "Virtual Prototyping이란?", 전자공학회지 제25권 제2호, 1998.

[7] 김철웅, 한상용, "DragonarVPSim : 가상 프로토타이핑 시뮬레이터", '99 정보처리학회 춘계 학술발표논문집 제6권 제1호, pp.955-958, April, 1999.

[8] R. K. Gupta, G. De Micheli, "Hardware-Software Cosyntesis for Digital Systems," IEEE Design & Test of Computers, 1993.

[9] W. Hardt, "An Automated Approach to HW/SW-Codesign," IEEE Colloguium, 1995.

[10] Harel David, "Statecharts : A Visual Formalism for Complex Systems," Sci. Computer, 1987.

[11] David Harel and Amnon Namad, "The Statemate Semantics of StateCharts" ACM Trans. On Software Engineering, 1996.

[12] David Harel, A.Pnueli, J.p.Schmidt, R. Sherman, "On the Formal Semantics of Statecharts," in Proc. 2nd IEEE Symp. Logic in Computer.

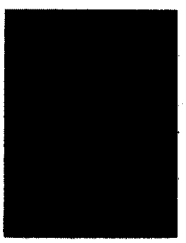
[13] B. J. Singh, "Prototyping preps for virtual approach,"

EETIMES, Sep, 1997.

[14] Rapid PLUS Case Studies, http://www.emultek.com/simulation/index.html.

[15] 김철웅, 한상용, "가상 프로토타입 명세를 위한 객체지향 기술 요소를 지닌 확장된 Statecharts", 정보통신 우수 대학원 발표 논문집, 중앙대학교 정보통신 연구소, pp.53-56, May, 1999.

[16] http://fundedresearch.cos.com/cgi-bin/feddb/NSF/nsf/150329508/150331304/projects/fundedresearch/data/nsf/nsf.data/seed=virtualprototyping.



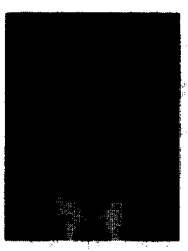
김 철 웅

e-mail : cukim@archi.cse.cau.ac.kr

1998년 중앙대학교 컴퓨터공학과 졸업(공학사)

1998년~현재 중앙대학교 컴퓨터공학과 석사과정

관심분야 : Virtual Prototyping, Embedded System, VLSI & CAD, Data backup, Security



한 상 용

e-mail : hansy@cau.ac.kr

1975년 서울대학교 공과 대학 졸업(학사)

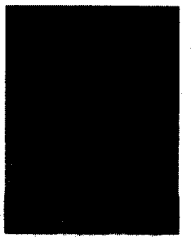
1977년~1978년 KIST 연구원

1984년 미네소타 대학교 컴퓨터공학과(공학박사)

1984년~1994년 IBM 연구소 연구원

1995년~현재 중앙대학교 컴퓨터공학과 교수

관심분야 : CAD, Virtual Prototyping, 망 접속



최 진 영

e-mail : choi@formal.korea.ac.kr

1982년 서울대학교 컴퓨터공학과(학사)

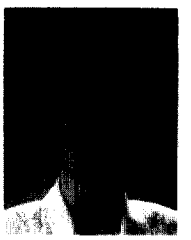
1986년 Drexel University 컴퓨터과학과(석사)

1993년 University of Pennsylvania 컴퓨터학과(박사)

1993년~1996년 University of Pennsylvania, research associate

1996~현재 고려대학교 컴퓨터학과 (부교수)

관심분야 : Computing Theory, Computational Logic, Real-Time Computing, Formal Methods (Formal Specification, Formal Verification, model checking), Programming Languages, Process Algebras, Software Engineering, and Protocol Engineering



이 정 아

e-mail : jeong@eunhasu.kjist.ac.kr

1982년 서울대학교 컴퓨터공학과(학사)

1985년 미국 인디애나 대학 컴퓨터학과(석사)

1990년 미국 UCLA 컴퓨터공학과(박사)

1990년~1995년 미국 휴스턴대학 전기전산공학과(조교수)

1993년~1994년 미국 국립 초전도가속기연구소(객원연구원)

1998년 TU, Delft, the Netherlands(초빙교수/연구원)

1995년~현재 조선대학교 컴퓨터공학부(부교수)

관심분야 : Computer Arithmetic, Application-specific Processor Design, Hardware/Software Co-Design, (Re)Configurable Computing