□기술해설□

# 실시간 소프트웨어 모델링[1]

고려대학교 전태웅*
경희대학교 이승룡**

목 차



## 1. 서 론

실시간 소프트웨어란 계산의 논리적 결과가 정확해야 할 뿐아니라 처리되는 결과들이 주어진 시간 내에 도출되어야만 하는 시간 제약이 따르는 소프트웨어이다. 실시간 성격을 갖는 소프트웨어는 로보트, 공정 처리, 발전, 플랜트, 빌딩/공장, 엘리베이터와 같은 산업용 전기전자 분야에서부터 통신, 교통 관제, 무기체계, 우주항공, 가전제품 분야에 이르기까지 그 응용분야가 매우 광범위하다. 오늘날, 실시간 소프트웨어가 내장된(embedded) 시스템과 이의 운용환경은 매우 복잡해지고 있으며 그러한 시스템에 대한 의존도도 가속적으로 심화되고 있다. 이에 따라 신뢰성 높은 실시간 소프트웨어의 효율적인 개발에 대한 중요성이 날로 높아지고 있다. 그러나, 실시간 소프트웨어 혹은 이를 내장한 실시간 시스템은 일반 소프트웨어와 구분되는 다음과 같은 특성들을 요구함으로써 그러한 시스템의 개발을 매우 어렵게 한다. [15, 26].

실시간 제약 : 실시간 시스템은 외부로부터의 입력 데이타나 사건(event)들의 처리를 주어진 시한내에 수행하여야 한다.

Reactive system : 대부분의 실시간 시스템들은 내부 및 외부환경으로부터 비동기적으로 발생하는 각종 자극들(stimuli)에 대해 끊임없이 반응하여야하는 사건구동적인(event-driven) 성격을 갖는다.

병행처리 : 외부 사건들은 동시 다발적이면서 비예측적으로 발생되는 경우가 많고 발생되는 사건들은 그 성격이나 유형에 따라 각각 요구되는 처리시한이 다르므로 이들에 대한 병행처리(concurrent processing)가 요구된다.

주기적 실행 : 미리 설정된 시각이나 시간 간격마다 주기적으로 수행해야 하는 처리 사항들이 있으며 이들 역시 시간 제약을 갖게 되는 경우가 많다.

*종신회원
**정 회 원

[1] 본 내용은 LG산전(실시간 소프트웨어 모델링 프로세스 정형화를 위한 연구 과제)과 한국과학재단('94 중점과제 연구회)의 지원을 받아서 작성됐음

내장 시스템 : 실시간 시스템은 일반적으로 보다 큰 시스템의 한 요소로서 내장된다. 예를 들면 로보트 제어기는 로보트 팔, 축 동작 제어를 위한 서보 메커니즘, 센서, 엑추에이터 등을 포함하는 로보트 시스템의 한 컴포넌트로서 내장된다.

고신뢰성 : 실시간 시스템은 매일 24시간 내내 작동이 요구되고, 요구된 기능이나 성능에 이상이 발생시 인명이나 재산상의 안전에 치명적인 손실을 초래하는 경우가 많아서 (safety-critical) 고도의 신뢰도를 유지해야한다.

실시간 소프트웨어의 위와같은 특성들은 소프트웨어의 기능과 구조를 매우 복잡하게 만들고 이의 분석, 설계 및 검증을 어렵게 하는 요인들이 된다. 이에따라 실시간 소프트웨어의 요구기능, 성능 및 설계 사양을 정확하고 명료하게 표현하고 분석, 이해, 검증할 수 있는 실시간 시스템 모델링 기법의 적용이 절실히 필요하다.

시스템 모델링이란 현존하는 시스템 혹은 구현해야할 시스템의 구조적, 동적인 특성들을 분석, 이해, 검증, 예측하기 위하여 시스템 구성요소 및 이들의 상호관계들을 도출, 표현하는 작업을 의미하는 광범위하고 일반적인 개념이다. 시스템 모델링은 공학 분야와 자연과학 분야에서는 이미 오래전부터 그 개념과 방법이 확고하게 정립되어 일상적으로 수행되어온 연구개발 방식이다. 현재는 CAD/CAE(Computer-Aided Design/Computer-Aided Engineering) 등과 같은 컴퓨터에 의한 모델링이 보편화되어 있으며, 소프트웨어의 개발에 있어서도 시스템 모델링의 개념과 방법을 도입, 적용하기위한 연구가 많이 진행되고 있다. 시스템 모델링과 밀접한 관계가 있는 명세화(specification), 분석(analysis), 설계(design), 시험(testing), 프로토타이핑(prototyping), 모의실험(simulation) 등의 개념들은 모두 소프트웨어의 개발에도 그대로 적용되며 넓은 의미에서 시스템 모델링의 범주에 포함된다.

소프트웨어 모델링의 목적은 이를테면 회로기판, IC 칩, 기계장치 등의 개발이 CAD에 의한 설계과정을 거쳐서 수행되는 것처럼, 소프트웨어의 개발시 CASE (Computer-Aided Software Engineering)에 의해 schematic diagram 등의 형태로 설계, 분석, 검증하는 과정을 거쳐서 수행하고자 함에 있다. 소프트웨어가 소스코드로 구현되기 전에 모델링 과정을 거침으로써 얻어지는 주요한 효과는 다음과 같다.

복잡도의 감소 : 소프트웨어의 제반 특성들이 상위 레벨에서의 핵심적인 사항들만 추상화되고 불필요하게 상세한 부분들이 은폐됨으로써 소프트웨어의 복잡도가 감소된다.

가시화 효과 : 소프트웨어의 전체적인 구조, 구성요소 및 구성요소들 사이의 관계들이 모델링 언어로 표현됨으로써 소프트웨어가 가시화된다.

이해도 향상 : 복잡도가 감소되고 주요 특성들이 가시화된 소프트웨어 모델은 구현될 소프트웨어에 대한 개발자와 사용자의 이해도를 향상시킨다.

예측 및 검증 : 소프트웨어 모델을 분석하고 실험, 관찰함으로써 소프트웨어에 요구되는 제반 특성들의 만족여부와 타당성을 조기에 예측하고 검증할 수 있다.

신뢰도 향상 : 소프트웨어에 내재될 수 있는 오류들이 모델링에 의해 미리 걸러진 후 소스코드로 구현됨에 따라서 소프트웨어에 요구되는 기능과 성능에 대한 신뢰도가 향상된다.

소프트웨어 모델링은 언어, 프로세스, 그리고 지원도구의 3가지 요소들로 구성된다. 모델링 언어는 소프트웨어의 핵심 사항들을 표현할 수 있는 어휘와 문법을 제공한다. 모델링 프로세스는 모델링 언어를 사용하여 소프트웨어를 모형화, 분석, 검증하는 기법과 절차를 의미한다. 모델링 지원도구는 소프트웨어 모델링의 수행 과정의 자동화를 지원하는 도구이다. 실시간 소프트웨어의 모델링은 지금까지 많은 연구가 진행되어 모델링 언어, 모델링 프로세스 및 이들에 대한 지원도구들이 많이 개발되어 왔다. 실시간 소프트웨어 모델링에 대한 최근의 주요 연구방향과 이슈들은 아래와 같다.

형식 모델(formal models) : 지금까지 문법(syntax)과 의미(semantics)가 엄밀하게 정의된 실시간 소프트웨어 모델링 언어들이 많이 소개되어 있으나 아직까지도 그 표현 범위와 적용가능한 응용 분야에 있어서 많은 제약을 가지고 있다. 정형화된 소프트웨어 모델링이 실

제 산업체 혹은 상업용의 대규모 실시간 소프트웨어에 확대(scale-up) 적용되기 위해서는 추상화(abstraction), 모듈화(modular structuring), 계층화(hierarchical & layered structuring) 등과 같은 시스템 구조화 메카니즘들이 보다 엄밀하게 모형화되어야 한다. 이러한 틀 안에서 실시간 제약조건과 같이 복잡하고 미묘한 사항들이 다양한 추상화 수준으로서 구조적으로 통합되어 표현될 수 있는 포괄적인 형식 모델링 언어에 대한 연구가 필요하다 [34].

실행가능한 모델(machine-executable models) : 복잡한 실시간 소프트웨어 모델을 효율적으로 분석, 검증하기 위해서는 소프트웨어 모델이 컴퓨터에 의해 실행가능해야 한다. 이러한 소프트웨어 모델링은 정형화된 모델링 언어의 사용이 필수적이다. 실행가능한 소프트웨어 모델은 구문 분석(syntax analysis), 정적 특성 분석(static analysis), 동적 특성 실행(dynamic execution), 모델 변환(model transformation), 시뮬레이션, 프로토타이핑 등을 컴퓨터에 의해 실행함으로써 구현될 소프트웨어에 요구되는 특성들을 미리 예측하고 검증할 수 있게 된다. 현재 머신 실행에 의하여 자동화된 형식 명세와 검증(automated formal specification & verification) 방법에 대한 연구가 많이 진행되고 있다 [15, 34].

시각적 모델(visual models) : 소프트웨어 모델이 그래픽의 형태로 표현되면 모델에 함축된 소프트웨어의 특성들이 시각적으로 부각되어 이해도가 향상되고 이에따라 모델의 분석과 검증이 용이해진다. 현재 시각화 모델을 형식화하기위한 시각적 정형화(visual formalism)와 이들의 자동화를 지원하는 도구와 개발환경에 대한 연구가 활발하게 진행되고 있다 [19, 34].

자동 생성과 합성(automated synthesis & generation) : 실시간 소프트웨어 모델은 궁극적으로 컴퓨터에 의해 실행가능한 소스 코드로 구현되어야 한다. 임의의 소프트웨어 모델로부터 소스코드 전부를 자동으로 합성, 생성하는 것은 거의 불가능하지만 정형화된 소프트웨어 모델로부터 소스코드를 부분적으로 자동 합성, 생성하기위한 연구는 많이 진행되어 있다 [2].

재사용가능한 모델(reusable models) : 실시간 소프트웨어 모델을 재사용가능한 형태로 만들고 실시간 소프트웨어 모델링을 기존의 유사한 모델들을 재사용하여 수행할 수 있으면 매우 효율적인 소프트웨어 모델링 프로세스가 될 수 있다. 소프트웨어 재사용의 핵심은 재사용도가 높은 패턴들을 찾아내어 이를 재사용가능한 형태로 가공하는 것이다. 그런데 소프트웨어 모델링은 하드웨어 모델링의 경우와는 달리 재사용도가 높은 표준 컴포넌트들과 디자인 패턴들을 찾아내는 것이 매우 어렵다. 현재 소프트웨어 재사용은 소스 코드 뿐만 아니라 설계나 요구분석 명세물들에 대해서도 그 적용이 가능한 방법들에 대한 연구가 활발하게 진행되고 있다 [2, 28].

과학적인 소프트웨어 모델링 기법의 적용은 시스템에 요구되는 기능 및 성능 조건들이 엄격하고 복잡한 실시간 소프트웨어의 개발에 특히 필요하나 실제 산업체에서 소프트웨어의 개발에 시스템 모델링 기법들이 적용되는 수준은 CAD/CAE의 사용이 보편화되어 있는 타 분야에 비하면 아직 초보적인 단계이다. 본 글에서는 실시간 소프트웨어의 개발에 적용가능한 주요 모델링 언어들과 모델링 프로세스들을 소개하고 간단한 실시간 소프트웨어인 자동차 속도제어기(cruise control system)를 모델링함으로써 실제 응용의 예를 보여준다.

## 2. 실시간 소프트웨어 모델링 언어

### 2.1 모델링 언어의 유형

실시간 시스템을 모델링하기 위해서는 먼저 모델의 구성 요소들을 표현하기위한 기본 어휘와 구문을 제공하는 모델링 언어 또는 표기(modeling language or notation)가 필요하다. 표현력이 풍부하고 문법(syntax)과 의미(semantics)가 잘 정의된 언어의 선택은 효과적인 시스템 모델링을 하는데 매우 중요하다. 모델링 언어는 언어의 표기 형태에 따라 텍스트 언어(textual language)와 시각 언어(visual language)의 두가지 유형으로 크게 나눌 수 있다. 텍스트 언어는 문자나 기호 등으로 구성된 문장이나 수식의 형식을 갖는 언어이다. 시각 언어는 다이어그램, 차트, 그래프 등과 같

은 그림의 형태를 갖는 언어이다. 텍스트 언어와 시각 언어는 상호보완적인 성격을 갖고 있다. 따라서 대부분의 시스템 모델링에 있어서 이들을 병행하여 사용하는 것이 효과적인 경우가 많다. 표 1은 실시간 소프트웨어의 모델링을 위하여 현재까지 개발된 대표적인 모델링 언어의 종류들을 위의 두가지 유형별로 분류한 것이다.

### 2.2 텍스트 모델링 언어

텍스트 모델링 언어는 그 형식이나 의미의 엄격성 정도에 따라 비정형(informal), 준정형(semi-formal), 그리고 정형(formal) 언어로 구분된다[34]. 자연어와 같은 비정형 언어는 이해하기 쉽고 표현력이 풍부하나 내용이 중복되고 모호하게 되는 단점이 있다. 의사코드(pseudocode)로 대표되는 준정형 언어는 자연어보다 훨씬 간결한 표현이 가능하고 논리의 제어 표현에 엄밀성을 갖고 있으나 그 외의 표현은 자연어와 같은 free format을 허용한다. 그 외의 텍스트 언어들은 형식이나 의미가 명확하게 정의된 formal notation들이다. 이 중 실시간 프로그래밍 언어[47]는 표현된 내용이 곧바로 컴퓨터로 실행가능한 장점이 있으나 너무 구체적으로 기술되는 단점이 있다. Mathematical equation/inequality, VDM [24], Z [46], real-time temporal logic(RTL) [31, 32], CSR [11], timed CSP(TCSP) [39] 등은 수학적인 기호와 논리에 의존하여 그 표현이 간결하면서도 엄밀한 반면 표기를 익히거나 이해하기가 어렵고 표현의 범위가 한정되어 있다.

위에서 언급된 정형 언어들 중 RTL, CSR, TCSP 등은 특히 타이밍, 처리시한, 타임아웃 등과 같은 실시간 시스템에 고유한 제약 조건들을 표현하는 수단들을 제공한다. 예를 들면, CSR은 자원(resources)이나 사건(events)의 처리와 관련된 시간 제약 조건들을 명시적으로 표현할 수 있는 언어이다.

텍스트 언어들 중 자연어나 의사코드는 지금까지도 소프트웨어 모델링에 가장 보편적으로 사용되고 있으며 정형 언어들은 표현의 엄밀성이 중요한 부분들에 대해 선별적으로 적용되고 있으나 실시간 소프트웨어와 같은 safety-critical system의 모델링에서는 정형 언어의 사용의 필요성과 비중이 점차 높아지고 있다.

표 1 실시간 소프트웨어 모델링 언어의 유형

| 표기 형태 | 개 요 | 예 |
|---|---|---|
| 텍스트 언어 (Textual languages) | 문자나 기호로 구성된 문장이나 수식의 형태 | ◆자연어<br>◆pseudocode<br>◆real-time programming language<br>◆set theoretic or predicate logic-based specification languages (예 : VDM, Z)<br>◆mathematical equation/inequality<br>◆real-time temporal logic (RTL)<br>◆communicating shared resources (CSR)<br>◆timed CSP (TCSP) |
| 시각 언어 (Visual languages) | 다이어그램, 차트, 그래프 등과 같은 그림의 형태 | ◆data/control flow diagram<br>◆entity-relationship diagram<br>◆state transition diagram<br>◆statechart<br>◆modechart<br>◆hierarchical multi-state machine (HMS)<br>◆extended state machine (ESM)<br>◆class/object diagram<br>◆structure chart<br>◆petri-net |

## 2.3 시각 모델링 언어

시각 모델링 언어는 나타내고자 하는 시스템의 구조나 특성을 그래픽을 통해 명료하면서도 입체적, 직관적으로 표현할 수 있는 장점으로

표 2 시각 모델링 언어의 종류

| 종 류 | 개 요 |
|---|---|
| Data/control flow diagram (DFD/CFD) | ◆실시간 구조적 분석/설계 방법론에서 사용<br>◆고전적인 구조적 방법론의 DFD를 실시간 응용에 맞게 확장<br>◆CFD는 데이타 흐름을 제어하는 제어(control)의 흐름 및 시퀀스를 표현 |
| Task structure diagram | ◆DARTS (Design Approach for Real-Time Systems) 설계 방법론에서 사용<br>◆시스템을 복수의 병행 태스크들로 분할하여 구성<br>◆태스크 사이의 인터페이스를 message, event signal, information hiding module 등의 형태로 나타냄 |
| MASCOT diagram [1, 45] | ◆시스템을 병행 태스크들로 분할하여 구성<br>◆태스크 사이의 인터페이스는 channel과 pool로 나타냄 |
| Structure graph | ◆시스템을 병렬 태스크, packages, procedures들로 구성<br>◆Ada 언어로 구현하기에 적합 |
| Structure chart | ◆구조적 설계 방법론에서 사용<br>◆시스템을 기능적으로 분할, calling hierarchy를 표현 |
| Entity structure diagram | ◆Jackson System Development (JSD) 방법론에서 사용<br>◆문제영역(problem domain)은 entity로, entity들 사이의 관계는 event로 표현 |
| JSD network diagram | ◆JSD 설계상의 process들과 이들 사이의 인터페이스를 표현<br>◆인터페이스는 data stream communication 혹은 state vector inspection의 형태로 표현 |
| Class diagram | ◆객체지향적 분석/설계 방법론에서 사용<br>◆시스템을 클래스들과 이들 상호간의 관계들로 표현 |
| Object diagram | ◆객체지향적 방법론에서 사용<br>◆시스템 내의 객체들 및 객체 상호간의 관계를 표현 |
| State diagram | ◆시스템의 상태 변환을 표현<br>◆실시간 구조적 방법론 및 DARTS 방법론에서 사용 |
| Statechart [16, 17] | ◆state diagram을 확장하여 계층, 병행, 연쇄반응의 표현기능을 추가<br>◆구조적 방법론이나 객체지향적 방법론에서 state diagram 대신 사용하거나 Harel의 Statemate 개발환경에서 시간이나 동적인 성질을 표현하는데 사용 |
| Modechart [23] | ◆state diagram에 이산형 시간에 대한 표현을 추가 확장<br>◆시간에 관련된 추론은 real-time logic의 표기형식을 사용<br>◆SARTOR의 front-end로서 사용 |
| Hierarchical multi-state machine (HMS) [10] | ◆petri-net, statechart, modechart, temporal logic 등의 표기형식을 이용한 상태 기반 수행 명세화 언어 |
| Communicating real-time state machine (CRSM) | ◆병행 실시간 시스템의 요구조건을 명세화하기 위한 상태전이 기반의 수행 표기<br>◆이산형 시간에 대해 병행성과 동기화 명세를 지원 |
| Extended state machine (ESM) [33] | ◆CSP-like I/O와 real-time temporal logic(RTL) 형식의 실시간 제약 표현방법을 추가하여 state diagram을 확장한 실시간 명세화 언어 |
| Petri-net [37] | ◆병행 시스템을 graph의 형식으로 모델링<br>◆조건(condition)과 전이(transition)의 2가지 유형의 node를 지원<br>◆상태 전이는 token의 이동으로 표현 |

인하여 지금까지 광범위하게 사용되고 있다. 시각 언어는 그 형식과 의미가 엄밀하게 정의되는 시각적 정형화로 점차 발전되고 있으며 하드웨어나 소프트웨어 기술이 뒷받침되어서 지금은 컴퓨터를 이용한 모델링, 분석, 시뮬레이션 등이 가능하다. 주요한 시각 모델링 언어들의 종류는 표 2와 같다. 표 2에서 DFD/CFD [20, 48], task structure diagram, MASCOT diagram [1, 45], structure chart, entity structure diagram [22], structure graph, JSD network diagram [22], class/object diagram [4]등은 주로 소프트웨어의 기능이나 구조를 모형화하는데 사용되며 state diagram, statechart[16, 17], modechart[23], CRSM [43], HMS [10], ESM [33], petri-net [29, 36, 37, 38, 40] 등은 상태 전이를 기반으로하여 소프트웨어의 동적 특성들을 모형화하는데 사용된다. 예를 들면, CRSM은 병행 실시간 시스템에서 상태전이의 수행 또는 동기화 시간을 명세화할 수 있는 표기를 제공한다.

시각 언어는 소프트웨어의 전체적인 구조나 동적인 특성들을 계층화하고 시각화하여 표현하는데 매우 편리하고 특히 상위 추상화 수준(higher abstraction level)의 특성들에 대한 모델링을 강력하게 지원하는 반면, 복잡한 실시간 제약 조건이나 하위 추상화 수준의 상세한 사항들의 표현 능력에는 한계가 있다. 이에따라 대부분의 시각 언어들이 텍스트 표기(textual notation)들을 부가하여 사용하는 것을 허용하고 있으며 소프트웨어 모델링은 시각 표기(visual notation)들을 주 표현 언어로 하고 텍스트 표기들은 필요에 따라 부수적으로 사용하는 방향으로 가고 있다.

## 3. 실시간 소프트웨어 모델링 프로세스

소프트웨어 모델링은 modeling notation을 잘 알고 있는 것만으로는 불충분하며 그러한 표기를 사용하여 실제로 모델을 구현해가는 과정이나 절차에 대한 체계적이고 구체적인 모델링 프로세스 혹은 모델링 방법론이 수반될 때 효과적인 적용이 가능하다.

### 3.1 모델링의 적용 단계

실시간 소프트웨어의 효과적인 모델링을 위해서는 먼저 체계적인 소프트웨어 개발 프로세스를 확립하고 이를 기본 틀로 하여 단계적으로 모델링을 해나가는 것이 중요하다. 소프트웨어 개발 프로세스는 크게 요구분석(requirements analysis), 설계, 구현, 시험, 그리고 유지보수(maintenance)의 5단계로 구분할 수 있다. 소프트웨어 모델링은 위의 소프트웨어 개발 프로세스의 모든 단계에서 적용이 가능하지만 특히 요구분석과 설계 단계에서 적용 효과가 크다. 그 이유는 초기단계에서의 엄밀하고 오류가 없는 모델링은 에러가 적은 시스템을 구축하여 유지보수의 비용을 줄일 수 있기 때문이다. 실시간 소프트웨어 모델링은 그것이 적용되는 소프트웨어 개발 단계에 따라서 표 3과 같이 environmental modeling, essential modeling, 그리고 implementation modeling의 3가지 유형으로 분류할 수 있다 [48].

먼저 요구분석 단계에서는 실시간 시스템의 외부환경 특성들과 요구 특성들을 분석, 모형화

표 3 적용되는 개발단계에 따른 모델링 유형

| 모델링 유형 | 적용 단계 | 목 적 |
|---|---|---|
| Environmental modeling | 요구분석 단계 | ◆시스템이 감시 혹은 제어할 외부환경의 특성들을 모델링<br>◆시스템 사용자의 운용환경 특성들을 모델링 |
| Essential modeling | 요구분석 단계 | ◆사용자나 외부환경의 관점에서 보았을 때 시스템에 요구되는 특성들을 모델링 |
| Implementation modeling | 설계 단계 | ◆설계자의 관점에서 본 시스템 내부 구조, 구성 요소, 구성 요소들의 기능, 동적 특성, 상호 관계 등을 모델링 |

하는 것이 필요하다. Environmental modeling과 essential modeling은 외부환경(controlled plants & human operators)과 실시간 소프트웨어(controllers)의 핵심적인 특성들을 각각 추상화하여 모델링하는 과정을 의미한다. 설계 단계에서의 implementation modeling은 실시간 소프트웨어에 요구되는 기능과 성능을 만족하는 시스템의 내부 기능, 구조, 동적 특성들을 도출, 모델링하는 과정이다.

### 3.2 모델의 표현 내용

실시간 소프트웨어는 앞에서 설명된 바와 같이 요구/제약 사항들의 성격이 다양하여서 이를 한가지 유형의 모델로 표현하는 것은 매우 어렵다. 이에따라 모델링 대상을 특정한 관점에 따라 그에 적합한 유형의 모델을 선택하여 복수의 모델들로 구성하는 것이 필요하다. 소프트웨어 모델은 표현되는 내용의 성격에 따라 각각 표 4와 같이 기능 모델(functional model), 구조 모델 (structural model), 그리고 행동 모델 (behavioral model)의 3가지 유형으로 구성된다 [18, 41].

기능 모델은 시스템에 요구되거나 필요한 기능 요소들과 이들의 상호 관계들을 기술하는 모델로서 데이타 변환(data transformations), 데이타 흐름(data flows), 데이타 저장소(data stores), 제어 변환(control transformations), 제어 흐름(control flows)등을 모형화한다. 구조 모델은 시스템의 구성 요소들과 이들 사이의 구조적, 기능적인 관계들을 기술하는 모델로서 태스크(tasks), 모듈(modules), 자료구조(data structures), 파일 구조(file structures) 등을 모형화한다. 행동 모델은 시스템에 요구되는 기능과 성능 조건을 만족하기 위하여 구성 요소들이 언제, 어떠한 조건하에서, 어떠한 순서로 수행되는가를 기술하는 모델로서 상태 전이, 타이밍, 제어 순서, 사건 발생 순서 등을 모형화한다.

위의 기능, 구조, 행동 모델은 서로 독립적이면서도 상호보완적인 관계를 지니고 있으며 실시간 소프트웨어 시스템은 위의 3가지 유형의 모델들이 전부 모아질 때 비로소 시스템의 전체적인 내용이 입체적으로 표현된다. 기능, 구조, 행동 모델은 요구분석과 설계의 어느 단계에서도 적용된다. 즉, environmental, essential, 그리고 implementation modeling 모두에게 적용할 수 있다.

### 3.3 시스템 분할 방식

소프트웨어 모델링에서 그 적용단계나 모형화의 대상에 관계없이 공통적으로 핵심이 되는 사항은 모델을 구성하는 시스템 컴포넌트들을 조직적으로 찾아내고 이에따라 시스템을 계층적으로 분할해가는 과정이다. 실시간 소프트웨어 모델링에서 시스템을 구성요소로 분할하는 기준이나 방식은 크게 다음의 3가지 유형으로 나눌 수 있다 [25].

1) 기능적 분할 방식 (Functional decomposition)
2) 데이타 중심 분할 방식 (Data-driven decomposition)
3) 객체지향적 분할 방식 (Object-oriented decomposition)

기능적 분할 방식은 데이타의 흐름, 변환, 저

표 4 표현된 내용의 성격에 따른 모델 유형

| 모델 유형 | 개 요 | 내 용 |
|---|---|---|
| 기능 모델<br>(Functional model) | 시스템에 외부적, 내부적으로 필요한 기능 요소들 및 이들 사이의 상호관계를 표현 | 데이타 유형, 데이타 변환, 데이타 흐름, 데이타 저장, 제어 흐름 |
| 구조 모델<br>(Structural model) | 시스템의 구조적인 특성들을 표현 | 프로세서, 태스크, 모듈, 데이타의 구성, 구조, 관계 |
| 행동 모델<br>(Behavioral model) | 시스템의 구성 요소들이 언제, 어떠한 순서로 수행하는 가와 같은 동적 특성들을 표현 | 이벤트 시퀀스, 상태 전이 시퀀스, 제어 시퀀스, 타이밍 |

장 등을 분석하여 시스템을 구성하는 세부 기능 요소들 및 이들 사이의 상호관계들을 data flow diagram, structure chart 등의 notation들을 사용하여 계층적으로 도출해나가는 방식으로서 Ward/Mellor나 Hatley의 구조적 방법론 등이 이에 속한다[20, 48]. 구조적 방법에 동적 특성을 계층적으로 표현할 수 있는 statechart[16]를 포함시킨 Statemate[18]는 시스템의 기능, 구조, 행동을 서로 동등한 수준에서 계층적으로 분할하는 모델링 프로세스를 지원한다. 데이타 중심의 분할 방식은 시스템에 요구되는 데이타의 유형 및 구조를 세분화하면서 분할하는 방식으로서 Jackson Structured Programming(JSP)[21]와 Warnier/Orr 방법론[30]이 이에 속한다. 객체지향적 분할 방식은 데이타와 데이타에 요구되는 오퍼레이션들을 분리하지 않고 하나의 객체 단위로 보고 그러한 객체 유형들을 계층적으로 도출해 나가는 방식으로서 Booch의 객체지향적 분석/설계 방법론 [3, 4]과 Rumbaugh의 Object Modeling Technique(OMT)[41] 등이 이에 속한다. 객체지향적 방식을 실시간 시스템의 모델링에 적합하도록 확장시킨 방법론으로서 Real-Time Object-Oriented Modeling(ROOM)[42]과 RTO. k 객체모델[27] 등이 있다. 위에서 언급된 방법들을 포함한 대표적인 실시간 모델링

표 5 대표적인 실시간 모델링 방법론

| Modeling method | 개 요 |
|---|---|
| Structured Analysis and Design for Real-Time Systems (RTSAD) [20, 44, 48] | ◆구조적 분석/설계 방법론을 실시간 시스템에 적용되도록 확장한 것<br>◆Ward/Mellor와 Hatley의 방법론이 가장 많이 사용됨 [20, 48]<br>◆Ward/Mellor와 Hatley의 방법론의 통합을 시도한 ESML (Extended System Modeling Language)가 있음<br>◆data flow diagram외에 control flow diagram과 state diagram이 주요 notation으로 사용됨 |
| Statemate [18] | ◆reactive system 개발에 적용하기 위해 Harel에의해 제안된 방법<br>◆executable specification과 visual formalism의 지원을 강조<br>◆기능, 구조, 동작 모델의 표현을 위하여 activity chart, module chart, statechart의 3가지 유형의 표기방식을 지원 |
| Naval Research Lab Software Cost Reduction Method (NRL) [35] | ◆onboard flight program과 같은 복잡한 실시간 시스템의 개발에 적용하기 위해 개발된 방법론<br>◆소프트웨어의 구조는 다음 3가지 독립적인 유형의 구조로 표현<br>: module structure, uses structure, process structure |
| Object-Oriented Analysis and Design [4, 8, 9, 41] | ◆abstraction과 information hiding의 개념에 의한 방법론<br>◆C++, Smalltalk와 같은 객체지향적 프로그래밍 언어로 구현될 경우 polymorphism과 inheritance와 같은 강력한 데이타 모델링을 지원<br>◆Rumbaugh의 OMT와 Booch의 방법론이 가장 보편적으로 적용됨<br>◆실시간 시스템 모델링을 위해 Selic에의해 확장된 Real-time Object-Oriented Modeling (ROOM) 기법 [42]과 K. H. Kim의 RTO. k object model [27] 등이 있음 |
| Jackson System Development (JSD) for Real-Time Systems [6, 7, 22] | ◆문제 영역 entity들의 behavior를 모델링<br>◆Jackson Structured Programming (JSP) [21]기법이 발전한 것 |
| Design Approach for Real-Time Systems (DARTS) [12, 13, 14] | ◆실시간 시스템을 다수의 병렬 태스크들로 구성하고 이들 사이의 인터페이스를 정의하는데 중점을 둔 방식<br>◆실시간 구조적 분석 사양으로부터 다수의 병렬 태스크를 도출하는 기준과 절차를 제시<br>◆DARTS의 방법에 따라 도출된 각각의 태스크의 세부 설계는 고전적인 구조적 설계 방식의 적용이 가능 |

방법들은 표 5에서 보여준다. 실시간 모델링 방법론들은 [15, 49]에서 잘 비교, 분석되어있다.

## 4. 실시간 소프트웨어 모델링의 예

본 장에서는 실시간 시스템 소프트웨어 개발시 소프트웨어 모델링이 구체적으로 어떻게 적용되는 지를 간단한 실시간 시스템의 예를 가지고 보여주고자 한다. 여기서 다룰 문제는 자동차의 자동속도 유지 장치인 cruise control system [3, 15]을 모델링하는 것이다.

### 4.1 문제 정의

시스템 모델링은 모델링 하고자하는 시스템의 성격, 기능, 작동환경 등 문제의 전반적인 윤곽을 서술함으로써 시작된다. 자동차 자동속도 유지장치(cruise control system) 문제는 다음과 같이 정의된다.

운전자가 가속기를 밟지 않고도 자동차의 속력을 일정한 속도로 유지하여 주는 기능이 있으면 교통이 한적한 도로를 장시간 운전하는데 매우 편리하다. 이러한 자동 속도 유지 기능을 cruise control이라고하며 이러한 기능을 수행하는 시스템을 개발하여 자동차에 장착시킬 수 있게하는 것이 주어진 문제의 목적이다.

Cruise control system은 작동시 현재의 속도를 감지하여 원하는 속도와 차이가 있을 경우 throttle을 조절하여 속도를 제어, 유지한다. cruise control은 엔진이 가동 중일 경우에만 작동한다. 엔진 시동시 cruise control system은 수동 모드인 off 상태로 된다. cruise control은 운전자가 브레이크를 밟거나 off를 명령할 경우 수동 모드로 전환된다.

### 4.2 요구분석 단계

Cruise control system 문제가 위와 같이 정의되면, 요구되는 기능 및 이에 파생되는 기능들을 보다 상세하고 명료하게 분석, 표현하는 작업이 필요하다. 이에 필요한 모델링 내용은 입출력 데이타, 데이타 흐름, 데이타 변환, 상태전이 등이 있다. 먼저 cruise control system과 외부 환경과의 경계를 이들 사이의 입출력 데이타와 함께 그림 1과 같이 블록 다이어그램으로 표현할 수 있다.

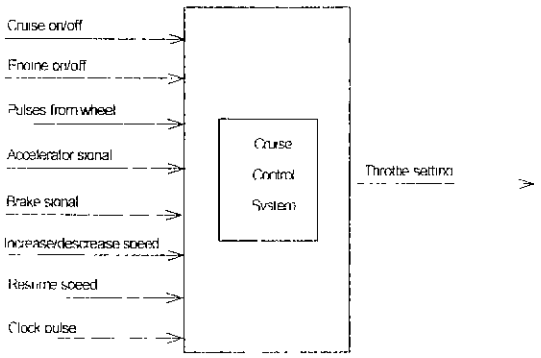



그림 1 Hardware Block Diagram for Cruise Control System

그림 1은 cruise control system에 입력이 필요한 데이타 유형과 외부로 출력될 데이타 유형을 보여줌으로써 시스템의 boundary를 설정하고 있다. cruise control system은 데이타의 입력 시점이 비동기적이고 데이타 입력시 그 유형이나 값에 따라서 처리 내용이 달라지는 event-driven 성격을 갖고 있다. 따라서 입력 데이타 유형 중 이벤트로서 다루어져야 할 것들을 찾아내고 각각의 이벤트들에 대해 시스템이 처리해야 할 내용을 명시하면 시스템을 분석하는 데 큰 도움이 된다. 표 6은 cruise control system의 외부 이벤트들 및 이들에 대한 시스템의 처리 내용들을 보여준다.

시스템의 입출력 boundary와 외부 이벤트들의 유형 및 이에 따르는 처리내용이 정리되면 이를 토대로 시스템이 내부적으로 어떠한 기능들을 수행해야 하는가를 data flow diagram (DFD)을 이용하여 표현, 분석할 수 있다. 그림 2는 cruise control system에 요구되는 세부기능을 보여주는 DFD이다.

그림 2에서 직사각형은 시스템 외부의 터미네이터들로서 시스템에 포함되지는 않으나 시스템과 입출력이 이루어지게 된다. 화살표는 데이타(혹은 이벤트)의 흐름을 나타내며 각이 둥근 도형은 데이타 변환(혹은 데이타 처리)을 나타낸다. 데이타 변환은 따라서 시스템 내부의 세부 기능을 표현한다. 한면이 열린 사각형은 데이타 저장소를 나타낸다. 그림 2는 따라서

표 6 Cruise Control System 이벤트 리스트

| 이벤트 | 처리 내용 |
|---|---|
| Cruise On/Off | On : 시스템에 의한 auto speed 모드로 변환한다.<br>Off : 운전자에 의한 manual speed 모드로 변환한다. |
| Engine On/Off | On : 자동차 엔진이 시동됨을 나타낸다.<br>Off . 자동차 엔진이 꺼짐을 의미한다. cruise control system의 작동을 중지한다. |
| Pulses from wheel | 자동차 바퀴의 1회전마다 pulse가 발생한다. pulse는 count되어 현재 속도를 계산하는데 사용된다. |
| Clock pulse | 1 msec 마다 clock pulse 발생. 현재 속도를 계산하는데 사용된다. |
| Accelerator signal | accelerator가 작동 중임을 나타낸다. |
| Brake signal | 운전자에 의해 브레이크가 작동 중임을 나타낸다. auto speed를 manual speed로 임시적으로 전환한다. |
| Increase/Decrease | 유지 속도의 가감 요청 발생을 나타낸다. 시스템이 off이면 무시된다. |
| Resume speed | auto speed로 전환하여 가장 최근의 유지 속도를 복원한다. 시스템이 off이면 무시된다. |

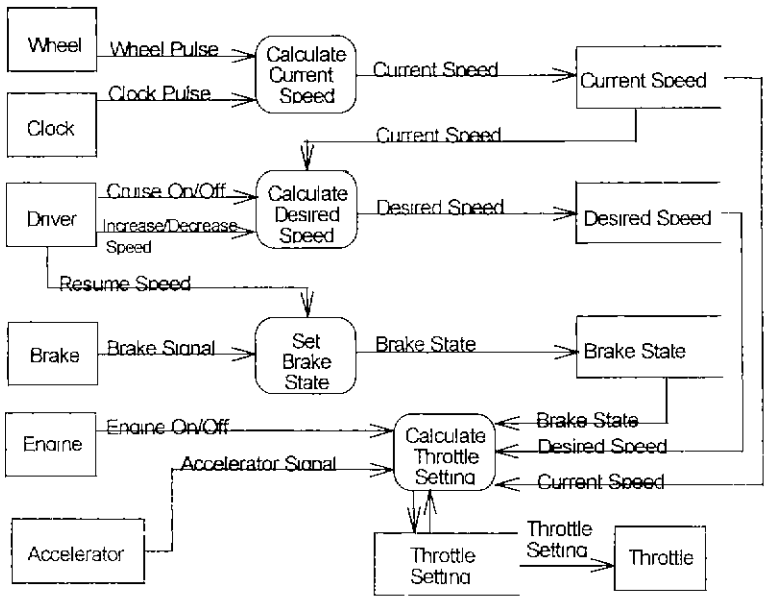



그림 2 Data Flow Diagram for Cruise Control System

cruise control system의 외부 및 내부에서 요구되는 데이타의 흐름, 변환, 및 저장소 등을 모델링함으로써 시스템에 요구되는 기능들을 간결하면서도 구체적으로 표현해 주고 있다. 그림 2는 그러나 특정 입력 데이타의 발생 시 이에 따른 데이타의 흐름, 변환, 저장, 출력이 어떠한 순서로 이루어지는가는 보여주지 않고 있다. cruise control system과 같은 실시간 시스템에서는 데이타의 처리 순서나 출력 시퀀스가 중요한 요구 사항이 되므로 이들을 요구분석

단계에서 모델링하는 것이 필요하다. 그림 3은 cruise control system에 요구되는 제어 시퀀스를 확장된 state diagram인 statechart로 모델링 한 내용을 보여준다.

그림 3에서와 같이 cruise control system은 engine on이거나 engine off인 두가지 상태 중의 하나에 있게된다. Engine on의 상태는 다시 on/off cruise, break on/off, accelerator on/off의 3가지 유형의 상태들의 조합으로 구성된다. Engine off에서 engine on으로 상태 전이가 발생하면 off-cruise, break off, accelerator off의 상태로 놓인다. off-cruise 상태는 cruise control system이 수동 모드에 있음을 나타낸다. off-cruise 상태는 다시 manual speed와 temporarily manual speed의 두가지 상태 중의 하나로 구분되며 off-cruise의 default는 manual speed이다. manual speed 상태에서 cruise on 이벤트가 발생하면 on-cruise 상태로 전이된다. 이 때, break나 accelerator의 상태는 변함이 없다. cruise control의 작동은 on-cruise 상태시에만 이루어진다. cruise control의 작동 상태는 그림에서와 같이 다시 maintaining speed, increasing speed, decreasing speed, resuming speed의 4가지 상태 중의 하나로 세분된다. 따라서 그림 3의 statechart는 cruise control system에서 요구되는 제어 시퀀스를 명료하면서도 입체적으로 표현하고 있다.

### 4.3 설계 단계

요구분석 단계에서 cruise control system의 기능과 동적인 요구 사항들이 모델링되면 이를 토대로 하여 구현될 시스템의 내부 구조를 결정하는 설계 단계로 넘어오게 된다. 시스템 설

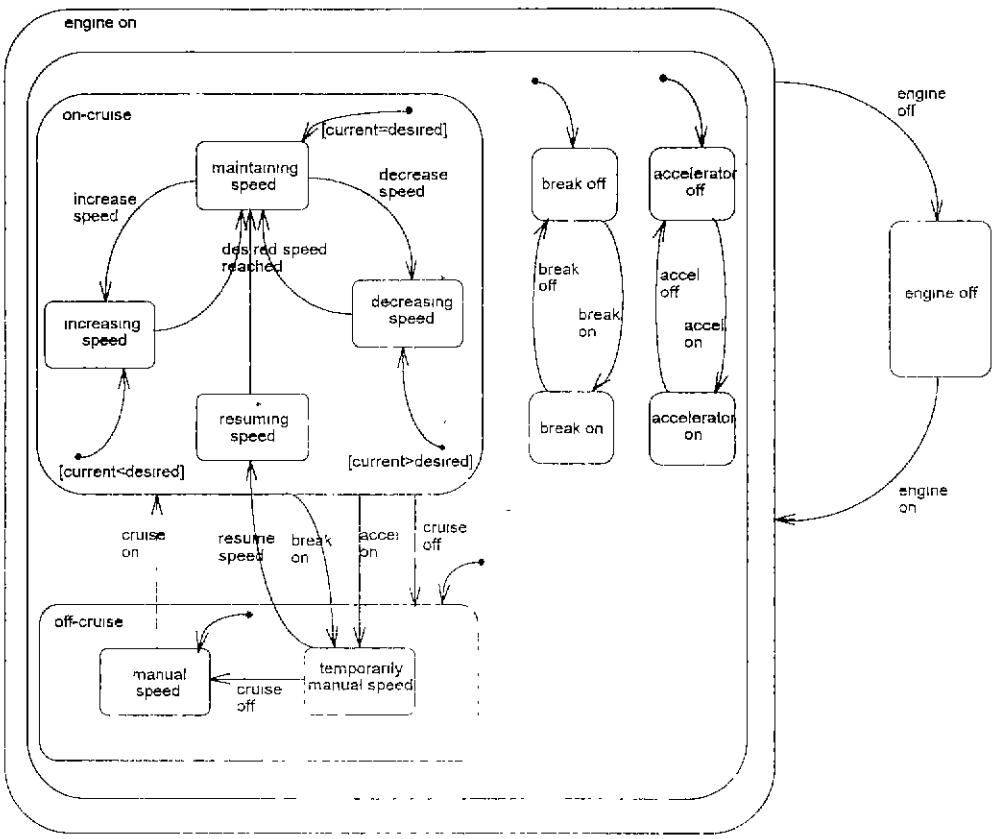



그림 3 Statechart for Cruise Control System

계는 시스템의 구성 요소들을 도출하고 이들의 결합 관계를 설정하는 시스템 분할(decomposition) 및 연결(interconnection & interfacing) 작업이 그 핵심이 된다. 시스템 분할은 기능을 중심으로 분할하는 기능적 분할(functional decomposition), 데이타를 중심으로 분할하는 데이타 분할(data-driven decomposition), 데이타와 데이타 오퍼레이션을 한개의 단위로 보는 객체지향적 분할(object-oriented decomposition)의 3가지 방식으로 분류할 수 있다. 여기서는 최상위 레벨에서의 기능적 분할과 객체지향적 분할의 적용을 각각 그림 4와 그림 5를 통하여 예시한다.

그림 4는 그림 2의 data flow diagram (DFD)으로부터 cruise control system을 기능적으로 분할하여 structure chart의 형태로 표현한 시스템 구조 모델이다. DFD로부터 structure chart를 도출하는 과정은 다음과 같다. 먼저 그림 2의 DFD에서 핵심이 되는 데이타 처리부분을 도출하고 그 데이타 처리에 필요한 입력 및 출력 부분들을 찾아내어 다음과 같이 구분한다.

데이타 처리 : calculate throttle setting
입력 데이타 : brake state, desired speed, current speed, engine on/off sig-

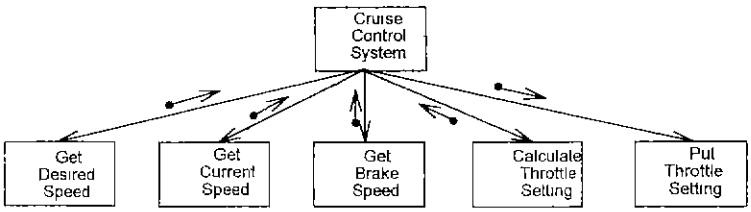



그림 4 Structure Chart for Cruise Control System (Functional Decomposition)

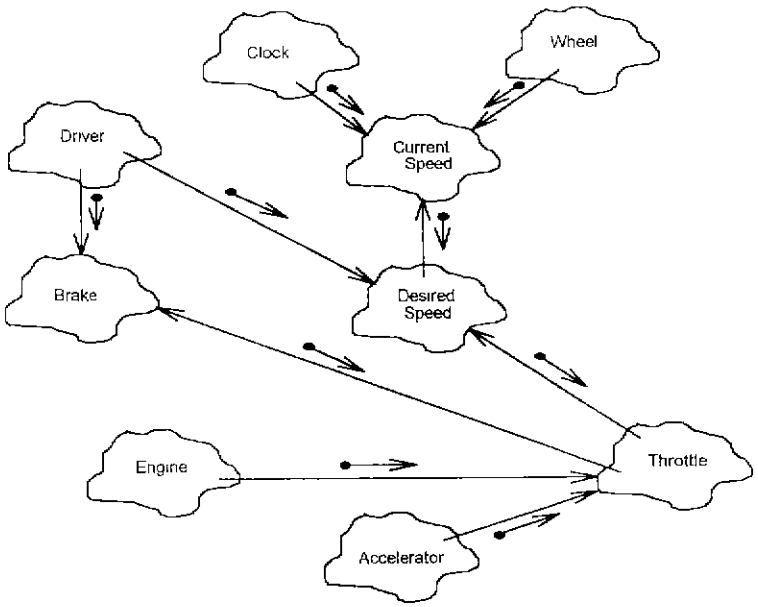



그림 5 Class Diagram for Cruise Control System (Obect-oriented Decomposition)

nal, accelerator signal
출력 데이타 : throttle setting 값

위의 내용을 바탕으로하여 입력 모듈, 처리 모듈, 출력 모듈들을 같은 레벨에서 도출하고 이들을 top level의 주모듈(main module)인 cruise control system과 연결한다. 이렇게 해서 나온 결과가 그림 4의 structure chart이다. 입력 데이타 중 engine signal과 accelerator signal은 Calculate Throttle Setting 모듈 내에서 입력되는 것으로 가정한다. 이 경우 Calculate Throttle Setting 모듈을 한 레벨 더 확장하면 engine signal과 accelerator signal의 입력 모듈이 Calculate Throttle Setting 모듈의 sub-module들로서 나타나게 된다. 그림 4에서 각 모듈을 연결하는 화살표는 모듈의 호출 방향을 표시하고 작은 화살표는 데이타 흐름의 방향을 표시한다. structure chart는 따라서 호출의 계층구조(calling hierarchy)를 나타내는 모델로도 볼 수 있다.

시스템 분할의 또다른 방식은 객체지향적 분할이다. 그림 5는 cruise control system을 객체지향적으로 분할하여 클래스 다이어그램의 형태로 모델링한 내용을 보여주고 있다. 그림 5도 기능적 분할과 마찬가지로 그림 2의 DFD로부터 도출된 것이다. 객체지향적 분할에서는 시스템 구성 요소들을 결정하는 기준이 데이타와 데이타에 대한 오퍼레이션들을 하나의 덩어리로서 갈라내는 것이다. 이러한 기준에 따라 DFD를 분석할 경우 터미네이터와 데이타 저장소가 주요한 구성 요소의 대상이 된다. 클래스 다이어그램 상의 구성 요소들은 클래스(혹은 객체)라고 불리운다. 그림 5의 클래스 다이어그램은 cruise control system이 크게 9개의 클래스들로 구성될 수 있음을 보여주고 있다. 이 중 current speed와 desired speed는 DFD의 데이타 저장소로부터 도출되었고 나머지는 터미네이터들로부터 도출된 클래스들이다. 각각의 클래스는 주어진 역할과 내부 데이타를 갖고 있으며 역할의 상당 부분은 다른 클래스들로부터 요청시 이를 서비스해주는 것이다. 그림 5에서 클래스들을 연결하는 화살표는 서비스 요청의 방향을 표시하고 작은 화살표는 데이타 흐름의 방향을 표시한다.

## 5. 결 론

소프트웨어의 개발에 있어서 기간과 비용의 초과, 개발된 소프트웨어의 기능과 성능의 불안은 오늘날의 소프트웨어 기술로서는 완전히 해결하기 어려운 문제로 남아 있다. 이는 소프트웨어가 지니고있는 복잡도, 가변성, 비가시성 등이 비규칙적이고 임의적이어서 공학적인 접근이 근본적으로 어렵기 때문이다 [5]. 소프트웨어 모델링은 소프트웨어의 개발을 보다 공학적인 접근방식에 의거하여 수행함으로써 소프트웨어에 내재된 이와 같은 어려움들을 극복하고자 함이다. 소프트웨어 모델링은 특히 실시간 소프트웨어와 같이 복잡한 시스템을 효과적으로 개발하는데 필수적으로 적용해야 할 매우 유용한 방식이다.

본 고에서는 현재까지 나와있는 실시간 소프트웨어 모델링의 개념, 유형, 주요 기법 및 실제 적용의 예를 살펴보았다. 실시간 소프트웨어는 그 개발에 있어서 고려되어야 할 사항들이 매우 다양하여 이들을 전부 한가지 모델로만 표현하는 것은 거의 불가능하고, 설사 하나의 모델로 표현이 가능할 지라도 그렇게 표현된 모델은 너무 복잡하여 이를 이해, 분석, 검증할 수가 없게 된다. 따라서 실시간 소프트웨어의 모델링은 시스템의 여러가지 측면을 몇개의 단면으로 나누어 각각에 적합한 모델을 선택하여 표현하는 것이 훨씬 효과적이다. 위에서 다룬 모델링 적용 사례는 요구분석 및 구조 설계 단계에서의 모델링 과정과 그 결과물들을 단순화하여 보여주고 있으며 후속 단계인 상세 설계, 코딩, 테스트 과정은 생략되어 있다. 요구분석이나 설계의 각 단계에 있어서도 실제의 복잡한 시스템에 있어서는 위의 예보다 훨씬 많은 양의 명세물들을 필요로 한다. 예를 들면 요구분석 단계에서의 자료 흐름(data flows)이나 상태 전이(state transitions)가 위와 같이 한 장의 그림으로 표현되는 경우는 극히 드물다. 소프트웨어가 이와같이 다양하고 방대한 내용과 형태로 모델링됨에 따라 다음과 같은 문제점들을 야기한다.

1) 모델의 복잡도가 증가하고 모델링된 명세물들 사이의 일관성과 traceability의 유지가 어렵다.
2) 모델링 과정이 복잡해지고 임의적으로 수행되어진다.
3) 모델에 오류가 내재될 가능성이 커지는 반면, 오류의 효과적인 발견이 어려워진다.
4) 모델의 신뢰도 보장 및 유지가 어려워지고 이에따라 향후 유사한 응용 분야의 소프트웨어 모델링에 재사용이 어려워진다.

실시간 소프트웨어 모델링에 있어서 위와 같은 어려움들을 해소하기 위한 연구가 현재 매우 활발히 진행되고 있다. 특히 실시간 소프트웨어를 보다 강력한 표현 능력을 지닌 정형화된 모델링 언어를 사용하여 컴퓨터에 의해 실행가능한 형태로 모형화하고 다양한 관점과 추상화 수준별로 모형화된 명세물들을 컴퓨터의 지원하에 분석, 실험, 검증, 변환 및 코드생성을 할 수 있는 포괄적인 모델링 언어, 모델링 프로세스, 모델링 지원도구에 대한 연구가 활발하다. 이러한 연구들이 계속 진행됨에 따라서 향후 실시간 소프트웨어 모델링은 점차 형식화, 시각화, 실행가능화, 재사용가능화한 방향으로 발전하여 [19] 실시간 소프트웨어의 효과적인 개발에 많은 도움을 줄 것으로 기대된다.

## 참고문헌


[1] S. T. Allworth and R. N. Zobel, Introduction to Real-Time Software Design, 2nd. Ed., Springer-Verlag, 1987.

[2] T. J. Biggerstaff and A. J. Perlis, Eds., Software Reusability : Volume I,II, ACM Press, 1989.

[3] G. Booch, Object-Oriented Development, IEEE Trans. Software Engineering, Vol. 12, pp. 211-221, Feb. 1986.

[4] G. Booch, Object-Oriented Analysis and Design, 2nd. Ed., Benjamin/Cummings Publishing Company, 1994.

[5] F. P. Brooks, "No Silver Bullet : Essence and Accidents of Software Engineering," IEEE Computer, Vol. 20, No. 4, pp. 10-19, Apr. 1987.

[6] J. Cameron, "An Overview of JSD," IEEE Trans. Software Engineering, Vol. 12, pp. 222-240, 1986.

[7] J. Cameron, JSP and JSD : The Jackson Approach to Software Development, 2nd. Ed., IEEE Computer Society Press, 1989.

[8] P. Coad and Ed Yourdon, Object-Oriented Analysis, 2nd. Ed., Prentice Hall, 1991.

[9] P. Coad and Ed Yourdon, Object-Oriented Design, Prentice Hall, 1991.

[10] A. Gabrielian and M. K. Franklin, "Multilevel Specification of Real-Time Systems," Communications of the ACM, Vol. 34, No. 5, pp. 50-60, May 1991.

[11] A. Gerber and I. Lee, "Communicating Shared Resources : A Model for Distributed Real-Time Systems," Communications of the ACM, Vol. 34, pp. 68-78, Dec. 1989.

[12] H. Gomaa, "A Software Design Method for Real-Time Systems, Communications of the ACM," July 1984.

[13] H. Gomaa, "Using the DARTS Software Design Method for Real-Time Systems," Proc. 12th. Structured Methods Conference, Chicago, Aug. 1987.

[14] H. Gomaa, "A Software Design Method for Distributed Real-Time Applications," The Journal of Systems and Software, Vol. 9, pp. 81-94, 1989.

[15] H. Gomaa, Software Design Methods for Concurrent and Real-Time Systems, Addison-Wesley, 1993.

[16] D. Harel, "Statecharts : A Visual Formalism for Complex Systems," Science of Computer Programming, Vol. 8, pp. 231-274, 1987.

[17] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman, "On the Formal Semantics of Statecharts," Proc. 2nd. IEEE Symp. Logic in Computer Science, New York, IEEE Press, pp. 54-64, 1987.

[18] D. Harel, et al., "STATEMATE : A Working Environment for the Development of Complex Reactive Systems," IEEE Trans. Software Engineering, Vol. 16, No. 4, pp. 403-414, Apr. 1990.

[19] D. Harel, "Biting the Silver Bullet : Towards a Brighter Future for System Development," IEEE Computer, Vol. 25., pp. 8-20, 1992.

[20] D. J. Hatley and I. A. Pirbhai, Strategies for Real-Time System Specification, Dorset House Publishing Company, 1988.

[21] M. Jackson, Principles of Program Design, Academic Press, New York, 1975.

[22] M. Jackson, System Development, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[23] F. Jahanian and A. K. Mok, "Modechart : A Specification Language for Real-Time Systems," IEEE Trans. Software Engineering, Vol. 20, No. 12, pp. 933-947, Dec. 1994.

[24] C. B. Jones, Systematic Software Development Using VDM, Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[25] K. M. Kavi and S. M. Yang, "Real-Time Systems Design Methodologies : An Introduction and a Survey," The Journal of Systems and Software, pp. 85-99, April 1992.

[26] K. M. Kavi, Ed., Real-Time Systems : Abstractions, Languages, and Design Methodologies, IEEE Computer Society Press, 1992.

[27] K. H. Kim and H. Kopetz, "A Real-Time Object Model RTO. k and an Experimental Investigation of its Potentials," Proc. COMPSAC'94, Taipei, pp. 392-402, Nov. 1994.

[28] C. W. Krueger, "Software Reuse," ACM Computing Surveys, Vol. 24, No. 2, pp. 131-183, June 1992.

[29] P. M. Merlin and A. Segall, "Recoverability of Communication Protocols-Implications of a Theoretical Study," IEEE Trans. Commun., pp. 1036-1043, 1976.

[30] K. Orr, Structured Systems Development, Prentice Hall, Englewood Cliffs, New Jersey, 1977.

[31] J. S. Ostroff and W. Wonham, "A Temporal Logic Approach to Real-Time Control," Proc. 24th. IEEE Conference on Decision and Control, pp. 656-657, 1985.

[32] J. S. Ostroff, "Real-Time Computer Control of Discrete Event Systems Modelled by Extended State Machines : A Temporal Logic Approach," Technical Report 8618, University of Toronto, Toronto, Canada, 1986, 1987.

[33] J. S. Ostroff and W. Wonham, "Modeling and Verifying Real-Time Embedded Computer Systems," Proc. IEEE Real-Time Systems Symposium, pp. 124-132, Dec. 1987.

[34] J. S. Ostroff, "Formal Methods for the Specification and Design of Real-Time Safety Critical Systems," The Journal of Systems and Software, pp. 33-60, Apr. 1992.

[35] D. P. Parnas, D. P. Clements, and D. Weiss, "The Modular Structure of Complex Systems," Proc. 7th. IEEE ICSE, Orlando, Florida, Mar. 1984.

[36] J. L. Peterson, "Petri nets," Computing Surveys, Vol. 9, No. 3, pp. 223-252, Sep. 1977.

[37] J. L. Peterson, Petri Net Theory and Modelling of Systems, Prentice Halls, Englewood Cliffs, New Jersey, 1981.

[38] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," Technical Report MAC TR 120, MIT, Cambridge, Massachusetts, 1974.

[39] G. M. Reed and A. W. Roscoe, "A Timed Model for Communicating Sequential Processes," Theoret. Comp. Sci., Vol. 58, pp. 249-261, 1988.

[40] W. Reisig, Petri Nets : An Introduction, Springer-Verlag, Berlin, 1985.

[41] J. Rumbaugh, M. Blaha, et al., Object-Oriented Modeling and Design, Prentice Hall, 1991.

[42] B. Selic, G. Gullekson, and P. Ward, Real-Time Object-Oriented Modeling, John Wiley & Sons, Inc., 1994.

[43] A. Shaw, "Communicating Real-Time State Machines," IEEE Trans. Software

Engineering, Vol. 18, No. 9, pp. 805-816, Sep. 1992.

[44] K. Shumate and M. Keller, Software Specification and Design : A Disciplined Approach for Real-Time Systems, John Wiley & Sons, Inc., 1992.

[45] H. Simpson and K. Jackson, "Process Synchronization in MASCOT," The Computer Journal, Vol. 17, No. 4, 1979.

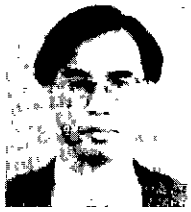
[46] J. M. Spivey, The Z Notation : A Reference Manual, Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[47] A. D. Stoyenko, "The Evolution and State-of-the-Art of Real-Time Languages," The Journal of Systems and Software, pp. 61-84, Apr. 1992.

[48] P. T. Ward and S. J. Mellor, Structured Development for Real-Time Systems, Prentice Hall, 1985.

[49] D. P. Wood and W. G. Wood, "Comparative Evaluations of Four Specification Methods for Real-Time Systems," Technical Report CMU/SEI-89-TR-36, Software Engineering Institute, Carnegie Mellon University, Dec. 1989.

**전 태 웅**



1981 서울대학교 계산통계학과 학사
1983 서울대학교 계산통계학과 석사
1992 Illinois Institute of Technology 전산학과 박사
1981~1983 금성통신(현 LG전자) 연구소 촉탁연구원
1983~1987 금성통신(현 LG전자) 연구소 주임연구원
1992~1995 LG산전 연구소 책임연구원
1993 한국과학기술원 정보및통신공학과 대우교수
1995~현재 고려대학교 전산학과 조교수
관심분야 : 실시간 소프트웨어 공학, 지식기반 소프트웨어 공학, 소프트웨어 테스트, 소프트웨어 재사용

**이 승 룡**

1978 고려대학교 공과대학 재료공학과 졸업
1987 Illinois Institute of Technology 전산학과 석사
1991 Illinois Institute of Technology 전산학과 박사
1990~1992 Governors State University 전임강사
1992~1993 Governors State University 조교수
1993~현재 경희대학교 전자계산공학과 조교수
관심분야 : 운영체제, 실시간 컴퓨팅, 멀티미디어 시스템