시스템 형식론에 의한 사용자 인터페이스 시스템 표현과 DEVS 모델링

System Theoretic Representation of UI System and DEVS Modeling

김은하* · 조대호*
Kim, Eun Ha and Cho,Tae Ho

Abstract

In this paper, we propose a software design method that will track the effects of modifications in a component to the rest of the components in the design phase. The prediction of the effects due to the design modifications before coding can be a valuable aid for the complex and large software development. Within the method, the target system is represented by the structured I/O system level specification which is one of the system representation level defined by the system theory. Then it is abstracted to the I/O system level. The DEVS (Discrete Event System Specification) model is constructed based on the I/O system level specification. Finally, the DEVS model is simulated to generate the behavior of the software by the abstract simulator in DEVS simulation environment. As an application, the graphic user interface system of a metal grating production scheduling system is presented.

Key Words: 시스템 형식론, DEVS, 사용자 인터페이스, 소프트웨어 변경관리, 상호 의존성 추적

* 성균관 대학교 전기전자 및 컴퓨터 공학부

1. 서 론

사용자 인터페이스는 사용자와 응용 시스템 간의 상호 대화를 위한 대화창구로서 사용자가 응용 시스템에 입력을 주고, 응용 프로그램이 사용자에게 결과를 보여준다. 소프트웨어 시스템의 기능이 복잡해지고, 사용자의 계층이 다양해짐에 따라, 사용자 인터페이스의 중요성이 크게 강조되고 있다. 그러나 소프트웨어 개발시 좋은 사용자 인터페이스를 설계하는 것은 어려운 작업이다. 인터페이스 소프트웨어는 많은 장치들을 빈번히 제어해야 하고, 비동기적으로 사건의 입력 스트림들을 보내야 하며 사용자의 행동과 시스템 반응 사이에 지체감이 없도록 보장해야 한다는 요구 사항을 만족해야 하는 특성 때문에 자주 프로토타입으로 만들어지고 반복적으로 수정되어야 한다[6]. 그러나 비교적 수행 절차에 필요한 알고리즘이 단순하고 처리 과정이 복잡하지 않으므로 사상 이론을 적용하여 자동 시스템을 구현하는 것이 용이하다.

그러므로 본 논문에서는 인터페이스 시스템의 설계 및 구현 과정에서 있을 수 있는 설계 변경 및 이에 따른 다른 변경 요인들을 정확하게 파악하고, 구현상의 변경으로 인한 전체 시스템의 영향 등을 체계적으로 정립하는 소프트웨어 설계 방법론을 특정 공정의 사용자 인터페이스 시스템을 통해 제시하였다.

2. 관련 연구

소프트웨어의 설계 방법론과 변경관리에 대한 연구가 활발히 진행되고 있다. 새로운 개념과 기법을 적용하여 기존의 변경관리 시스템의 문제점들을 보안한 몇가지 변경관리 시스템들을 살펴보면 다음과 같다.

첫째는, 의미론적 데이터모형과 메타모형의 유용한 개념을 추가한, 확장된 객체지향 데이터모형을 이용한 변경관리 시스템[7]은 개체간의 상호의존성에 함축된 의미들을 몇가지의 요소관계성들로 추상화함으로써 다양한 관계성들을 데이터모형 내에 경제적으로 정의하여 광범위한 변경관리가 가능하게 하였다. (1) 제한된 숫자의 관계성만을 다루기 때문에 변경 처리 범위가 국부적이거나 (2) 관계성의 의미를 시스템 내부에 고착시키기 때문에 다양한 사용자 환경에 적응하기 어렵다는 문제점을 갖는 기존의 변경관리 시스템들을 보안하였으나, 요소 관계성이나 변경 규칙 등을 알아야 한다는 어려움이 있다.

둘째는, MVCD(model-view-controller-dialogue)[6]는 MVC(model-view-controller) 모델이 갖는 강한 결합력으로 인한 소프트웨어 재사용의 저하를 해결하기 위해, 상호 작용의 구문적 관리를 하는 다이알로그 객체를 추가하여 사용자로부터 생성된 메시지가 컨트롤러에 의해 바로 값이 변경되지 않고 다이알로그에 전달되도록 하였다. 이러한 사용자 인터페이스 개발 도구를 사용하면 소프트웨어 개발시 사용자 인터페이스를 구축하는데 불필요하게 낭비되는 시간과 노력을 절감할 수 있다는 장점이 있으나, 사용자 인터페이스라는 제한된 도메인의 개발만을 지원하고 있다는 한계를 갖는다.

본 논문에서는 시스템 형식론을 적용하여 인터페이스 시스템을 입출력 시스템, 구조적 입출력 시스템, DEVS 모델로 재구성함으로써 추상화된 DEVS 모델을 통해 인터페이스 시스템에서의 변경이 전체 시스템에 미치는 영향을 체계적으로 정립하는 소프트웨어 설계 방법론을 제안하였다.

3. 배경 이론

이 장에서는 시스템 명세의 계층 구조를 설계하는데 적용되는 시스템 이론, DEVS 방법론, 객체지향 프로그래밍에 대한 기본 개념을 살펴본다.

3.1 시스템 이론 (System Theory)

시스템 이론에서 정의하는 시스템 형식론은 계층적 구조를 갖는 시스템 명세(System

Specification)와 각 계층의 시스템간의 구조 관계(Interrelation)에 대해 정의한다. 시스템 명세는 입출력에 의한 동적 특성만을 파악하는 하위 계층에서부터 형식적인 구조를 갖는 상위 계층으로 구성되어 있다. 시스템간의 구조 관계는 같은 계층에서 한 시스템과 다른 시스템과의 관계성을 명시한다. 상위 계층의 구조 관계는 하위 계층의 구조 관계를 포함한다. 즉, 상위 계층은 하위 계층의 모든 구조적인 특징을 포함한다[1,2]. <표 1>은 계층성을 갖는 시스템의 종류와 형식을 나타낸다. 5계층의 시스템 명세와 각 시스템간의 구조 관계에 대한 정의는 다음과 같다. 다음의 5계층의 시스템 중에서 본 연구에서는 계층 2,3의 입출력 시스템과 구조적 입출력 시스템을 적용하여 설계하였다.

I/O Relation Observation(IORO)은 시스템의 입력이 되는 X의 집합과 Y의 집합으로 구성되고, 입력 세그먼트 (X,T)에 의한 출력 세그먼트 (Y,T)와의 관계만을 명시한다.

I/O Function Observation(IOFO)은 입력 세그먼트과 그에 따른 출력 세그먼트의 모임을 입출력 함수의 집합으로 표현한다.

IORO(T,X,Ω,Y,R)와 IOFO(T,X,Ω,Y,F)의 구조 관계는

$$R = \cup_{f \in F} f$$

즉, 입출력 관계를 추상화시켜서 함수의 모임으로 표현한다.

I/O System Specification은 연속적인 입력 세그먼트의 결합(Composition)된 형태의 세그먼트 집합을 고려한 시스템이다.

시스템 $S=<T,X,\Omega,Q,Y,\delta,\lambda>$에서 각 상태 $q \in Q$마다 $B_q:\Omega \rightarrow (Y,T), \omega \in \Omega$의 관계가 있고, 이를 상태 q에 대한 입출력 함수(I/O function)라 한다.

$$\beta_q(\omega) = OTRAJ_{q,\omega}$$

IOFO (T,X,Ω,Y,F)와 $S = <T,X,\Omega,Q,Y,\delta,\lambda>$의 구조 관계 $B_S=\{\beta_q | q \in Q\}$는 I/O behavior로서 $IOFO_S$ (T,X,Ω,Y,B_S)라고 표현 할 수 있다.

Structured I/O System Specification는 추상화된 집합과 함수들을 원시 집합과 함수들의 크로스 프로덕트(cross product)로써 표현한다.

$M = <Q,\delta>$은 다음과 같은 구조를 갖는다.

D (coordinates)

$\{Q_\alpha | \alpha \in D\}$ (range sets)

$\{I_\alpha | \alpha \in D, I_\alpha \subseteq D\}$ (influencers),

$\{\delta_\alpha | \alpha \in D, \delta_\alpha : \times_{\beta \in I_\alpha} Q_\beta \rightarrow Q_\alpha\}$

(local transition functions)

$S = <T,X,\Omega,Q,Y,\delta,\lambda>$와 $M = <Q,\delta>$의 구조 관계 다음과 같이 정의된다.

$$Q \subseteq \times_{\alpha \in D} Q_\alpha$$

$$\delta = \times_{\alpha \in D} \delta_\alpha \cdot projI_\alpha \text{ restricted to } Q$$

<표 1> 시스템 명세의 계층도[1,2]

계층	시스템 명세
0	I/O Relation Observation
1	I/O Function Observation
2	I/O System Specification
3	Structured I/O System Specification
4	Network of Specifications

3.2 DEVS 방법론

본 논문에서의 시뮬레이션 모델링은 Zeigler에 의해 정립되어 이산사건 모델들의 계층 구조적 모듈화 방법을 제공해주는 형식론인 DEVS (Discrete Event System Specification)에 따른다. DEVS는 시스템이 일반적으로 갖는 특성들을 정의하여 시스템을 모델링할 수 있는 프레임웍(Framework)을 제공하며 형식론을 채택하여 시스템을 관찰하고자 하는 초점으로 추상화하여 모델링함으로써 모델과 실제 문제와의 관련성을 높일 수 있다. DEVS의 기본(Basic) 모델은 다음과 같은 항들로 명세할 수 있다[3,4,5] :

$$M = <X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta>$$

X : 입력 사건(Event)들의 집합

S : 사건의 변화에 따라 가질 수 있는 상태들의 집합

Y : 출력 사건들의 집합

δ_{int} : $S \rightarrow S$, 내부 상태전이(State transition) 함수

δ_{ext} : $Q \times X \rightarrow S$, 외부 상태전이 함수

λ: $S \rightarrow Y$, 출력 함수

ta : $S \rightarrow R^{+\infty}_0$, 시간 진행 함수

where Q = {(s,e) | s∈S, 0≤e≤ta(s)}
e: 최근의 상태 전이 이후로 경과된 시간

3.3 객체지향 프로그래밍

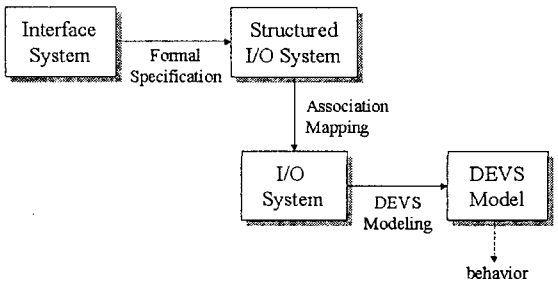
객체지향 프로그래밍은 캡슐화를 통하여 소프트 웨어의 복잡도를 제어하는데 도움이 되는 패러다임을 제공하기 때문에 사용자 인터페이스 개발에 매우 중요하다. 객체지향 프로그래밍은 상속에 의해 기존의 소프트웨어의 재사용을 증진시킨다. 객체지향에 의해 제공되는 기존의 부품을 수정하고 재사용하는 능력은 광범위한 프로그래밍 없이 사용자 인터페이스에 대한 프로토타입을 생성하는 것을 가능하게 한다. 객체지향 프로그래밍은 동일한 속성, 동일한 행위, 동일한 객체들과의 동일한 관련성, 동일한 시멘틱을 갖는 객체 집합에 대한 일반적인 기술체로써 클래스를 정의한다.

클래스내의 모든 객체들은 속성의 값만을 다르게 가질 뿐 동일한 속성과 행위를 갖게 되므로 이러한 개념으로 구성된 인터페이스 시스템의 각 클래스는 구조 관계에 의해 구조적 입출력 시스템의 구성 요소(coordinate)들로 변환된다. 그러므로 객체지향 프로그래밍 사고틀에 의해 인터페이스 시스템을 구현하는 것은 매우 중요한 의미를 갖는다[6,8,12,16,18].

4. 시스템 명세의 계층 구조

인터페이스 시스템을 <그림 1>과 같이 시스템 이론에서 정의하는 구조적 입출력 시스템 레벨(Structured I/O System Level)의 요소들로 표현하고 다시 구조적 입출력 시스템 레벨을 입출력 레벨(I/O System)로 변환한다. 이를 다시 DEVS 모델로 재구성하여 DEVS 시뮬레이션 환경에서 제공하는 시뮬레이터를 통하여 대상 시스템의 중요한 동적 특성을 소프트웨어 설계 시 또는 설계 변경 후 미리 파악할 수 있도록 한다.

4.1절에서 4가지 시스템의 명세를 살펴보고, 4.2절에서는 각 시스템간의 변환 과정을 보여주는 구조 관계를 살펴보도록 한다.



<그림 1> 시스템 명세와 구조 관계

<표 2> 계층적 구조 관계의 예

DEVS 모델	입출력 시스템	구조적 입출력 시스템	인터페이스 시스템
TGA-1	q_1	P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-1}	CUIApp.InitInstance()
TGA-2	q_2	P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-2}	CUIApp.InitInstance()
TGA-3	q_3	P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-3}	CUIApp.InitInstance()
TGA-4	q_4	P_{01-2},P_{02-1},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-4}	COrderBriefView.OnOK() CUIApp.InitInstance()
TGA-5	q_5	P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-25}	COrderBriefView.OnInit() CUIApp.OnInitialize()

<그림 1>과 같은 4가지 시스템들의 사상을 통해 <표 2>와 같은 상호 의존성을 추적하고, 이를 통해 인터페이스 시스템의 변경관리를 하는 것이 시스템 설계의 목표이다.

예를 들어 <표 2>에서 DEVS 모델의 상태가 'TGA-5'이면, 입출력 시스템의 상태는 'q_5', 구조적 입출력 시스템의 각 컴포넌트의 상태는 'P_{01-1}, P_{02-1}, P_{03-1}, P_{04-1}, P_{05-1}, P_{06-1}, P_{07-1}, P_{08-1}, P_{09-25}'이고, 이때 인터페이스 시스템의 클래스는 COrderBriefView와 CUIApp임을 알 수 있다.

4.1 시스템 명세

본 논문에서는 시스템 명세를 인터페이스 시스템, 구조적 입출력 시스템, 입출력 시스템,

DEVS 모델 등 4가지 시스템으로 구분하였다. 각각의 내용과 적용된 예를 살펴보면 다음과 같다.

4.1.1 인터페이스 시스템 (Interface System)

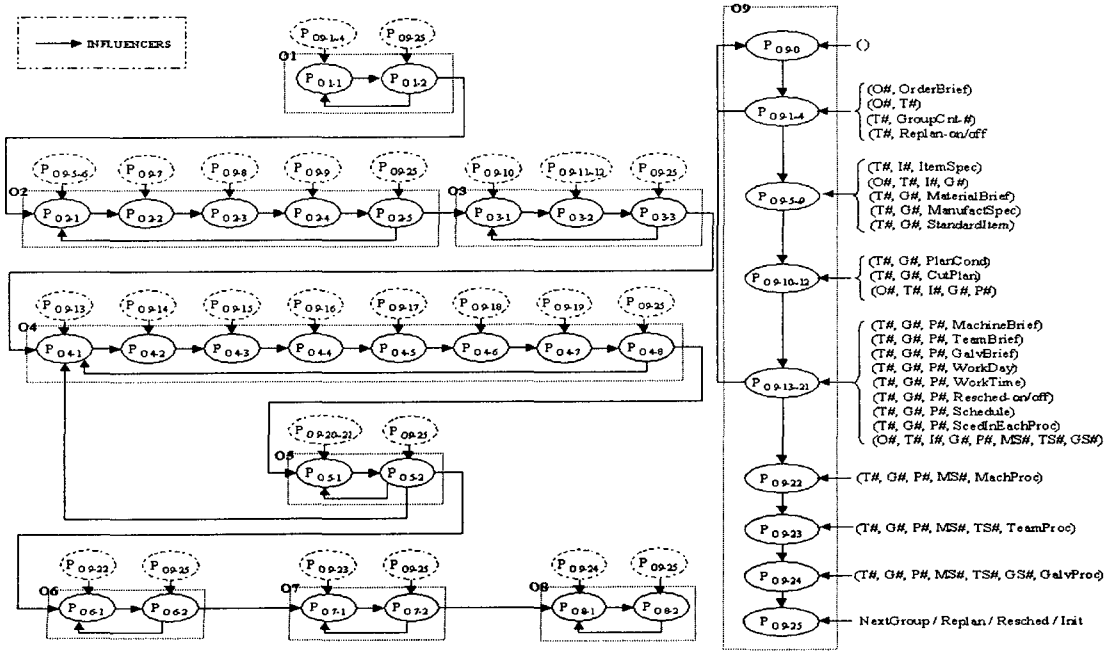
인터페이스 시스템은 소프트웨어 공학적인 토대와 객체지향 프로그래밍 사고 틀에 의해 Visual C++, SpreadSheet, ChartFX등과 같은 툴을 이용해서 구성한 특정공정의 인터페이스 시스템이다.인터페이스 시스템은 <표 3>과 같은 클래스와 멤버함수로 구성되어 있다.

4.1.2 구조적 입출력 시스템 (Structured I/O System)

인터페이스 시스템을 <표 4>와 같이 원시 집합과 함수들의 크로스 프로덕트(cross product)로써 표현함으로써 구조적인 시스템을 구성한다. 즉, 인터페이스 시스템을 컴포넌트간의 연관

<표 3> 인터페이스 시스템의 클래스와 멤버함수

클래스	멤버함수 (생성자, 소멸자 함수는 생략)
CUIApp	InitInstance(), OnInstance(), ...
COrderBriefView	OnInitialUpdate(), OnDblSpread(), OnEdit(), OnDelete(), OnOK(), ...
CPlanItemSpecView	OnInitialUpdate(), OnDblSpread(), OnRefresh(), OnEdit(), OnOK(), ...
CPlanGenrView	OnInitialUpdate(), OnGenPlan(), OnDelPlan(), OnViewPlan(), ...
CSchedGenrView	OnInitialUpdate(), OnGenrSched(), OnInitState(), OnOK(), ...
CSchedEachProcView	OnInitialUpdate(), OnOK(), ...
...	...



<그림 2> 구조적 입출력 시스템의 구성 요소간의 영향도

관계, 영향도 등으로 표현한다[1,2].

구조적 입출력 시스템은 $(D,\{Q_\alpha\},\{I_\alpha\},\{\delta_\alpha\})$와 같은 구조를 갖는다.

- D는 인터페이스 시스템의 클래스에서 생성된 오브젝트(구성요소)들로써 D={O1, O2,O3, ...,O9}와 같이 9개로 구성된다.
- Q_α는 오브젝트가 갖는 상태 집합으로써 $\{Q_\alpha\}=\{P_{O1},P_{O2},P_{O3},..,P_{O9}\}$와 같다. 예를 들어, 오브젝트 O1의 상태 변수 P_{O1}는 $\{P_{O1-1},P_{O1-2}\}$ 작업 절차에 따른 2개의 상태로, O2의 P_{O2}는 $\{P_{O2-1}, P_{O2-2}, P_{O2-3},P_{O2-4},P_{O2-5}\}$ 5개의 상태로 구성된다.
- I_α는 각 오브젝트에 영향을 주는 요소(Influencer)의 집합으로써 $\{I_\alpha\}=\{I_{O1},I_{O2}, I_{O3},...,I_{O9}\}$로 구성된다. 예를 들어, 오브젝트 O1에 영향을 준 Influencer는 {O1,O9}이고, O2에 영향을 준 Influencer는 {O1,O2,O9}이다.
- δ_α는 I_α의 영향을 받아 상태전이를 발생시키는 함수로써 $\{\delta_\alpha\}=\{\delta_{O1},\delta_{O2},\delta_{O3},...,\delta_{O9}\}$와 같다. 예를 들어, 오브젝트 O1의 상태 P_{O1-1}는 $\{P_{O9-1}, P_{O9-2},P_{O9-3},P_{O9-4}\}$의 영향을 받아 P_{O1-2}로 상태 전이되고, P_{O1-2}는 $\{P_{O9-25}\}$의 영향을 받아 P_{O1-1}로 상태 전이된다.

<그림 2>를 보면 9개의 오브젝트(D)로 구성되어 있고, 각 오브젝트는 여러 개의 상태(Q_α)로 구성되어 있다. 오브젝트의 여러 단계의 상태에 영향을 주는 요소(I_α)는 그림에서 화살표로 표현되어 있다. 또한 여러 개의 Influencer의 영향을 받아 로컬 상태전이 함수(δ_α)에 의해 상태전이를 하게 된다.

<표 4> 구조적 입출력시스템의 정의 및 적용예

정의	M (machine) = $< Q, \delta >$ where Q : the cross product of the component states sets δ : the cross product of component functions
정의	structure of M : $(D,\{Q_\alpha\},\{I_\alpha\},\{\delta_\alpha\})$ where D (coordinates) $\{Q_\alpha \mid \alpha \in D\}$ (range sets) $\{I_\alpha \mid \alpha \in D,\ I_\alpha \subseteq D\}$ (influencers) $\{\delta_\alpha \mid \alpha \in D, \delta_\alpha : \times_{\beta \in I_\alpha} Q_\beta \rightarrow Q_\alpha\}$ (local transition function) such that $Q \subseteq \times_{\alpha \in D} Q_\alpha$, and $\delta = \times_{\alpha \in D} \delta_\alpha$ ° proj I_α restricted to Q
적용	M = $< Q, \delta >$ where Q = { P_{O1}, P_{O2}, P_{O3}, ... , P_{O9} } δ = { δ_{O1}, δ_{O2}, δ_{O3}, ... , δ_{O9} }
적용	structure of M : $(D,\{Q_\alpha\},\{I_\alpha\},\{\delta_\alpha\})$ where D = { O1, O2, O3, ... , O9 } $\{Q_\alpha\}$ = { P_{O1}, P_{O2}, P_{O3}, ... , P_{O9} } such that P_{O1} = (P_{O1-1}, P_{O1-2}), P_{O2} = (P_{O2-1}, P_{O2-2}, P_{O2-3}, P_{O2-4}, P_{O2-5}), P_{O3} = (P_{O3-1}, P_{O3-2}, P_{O3-3}), ... , and P_{O9} = (P_{O9-0}, P_{O9-1}, P_{O9-2}, ... , P_{O9-25}) $\{I_\alpha\}$ = { I_{O1}, I_{O2}, I_{O3} , ... , I_{O9} } such that I_{O1} = {O1,O9}, I_{O2} = {O1,O2,O9}, I_{O3} = {O2,O3,O9}, ... , and I_{O9} = {O9} $\{\delta_\alpha\}$ = { δ_{O1}, δ_{O2}, δ_{O3}, ... , δ_{O9} } such that δ_{O1} : $P_{O1} \times P_{O9} \rightarrow P_{O1}$, δ_{O2} : $P_{O1} \times P_{O2} \times P_{O9} \rightarrow P_{O2}$, δ_{O3} : $P_{O2} \times P_{O3} \times P_{O9} \rightarrow P_{O3}$, ... , and δ_{O9} : $P_{O9} \rightarrow P_{O3}$

4.1.3 입출력 시스템 (I/O System)

<표 5>과 같이 입력 세그먼트에 의한 상태전이와 특정 상태에서 발생하는 출력 함수 등의 추상화된 집합과 함수로 구성된 시스템이다[1,2].

입출력 시스템은 $<T,X,\Omega,Q,Y,\delta,\lambda>$와 같은 표현식을 갖는다.

- T는 Time base로써 이산적인 사건의 흐름을 나타내는 정수(Integer)를 단위로 한다.
- X는 입출력 시스템의 입력 집합으로써 X ={(O#,OrderBrief),(O#,T#),(T#,GroupCnt-#),...} 등과 같은 입력값으로 구성되어 있다.
- Ω는 입력 세그먼트 집합으로써 observation interval <0,1>에서 입력 X가 발생하며, $\Omega=\{\omega \mid \omega:<0,1>\rightarrow X\}$와 같다.
- Q는 상태 집합으로써 $Q=\{q_0,q_1,q_2,q_3,...,q_{31}\}$로 구성된다. 각 상태는 구조적 입출력 시스템을 구성하는 9개의 오브젝트의 상태 변수로 표현된다. 예를 들어, q_0일 때 9개의 오브젝트의 상태는 $(P_{O1-1},P_{O2-1},P_{O3-1},P_{O4-1},P_{O5-1},P_{O6-1},P_{O7-1},P_{O8-1},P_{O9-0})$이고, q_1일 때 9개의 오브젝트의 상태는 $(P_{O1-1}, P_{O2-1},P_{O3-1},P_{O4-1},P_{O5-1},P_{O6-1},P_{O7-1},P_{O8-1},P_{O9-1})$이다.
- Y는 출력 집합으로써 입력 집합과 동일하다.
- δ는 $\delta : Q\times\Omega\rightarrow Q$와 같이 입력 세그먼트

<표 5> 입출력 시스템의 정의 및 적용 예

정의	S (system) = < T,X,Ω,Q,Y, δ , λ > where T: time base X : the input value set Ω: the input segment set (a subset of (X,T)) Q: the state set Y: the output value set δ : the state transition function λ : the output function
적용	S = < T,X,Ω,Q,Y, δ , λ > where T = Integer X = {(O#,OrderBrief),(O#,T#),(T#,GroupCnt-#), (T#,I#,ItemSpec),(O#,T#,I#,G#),(T#,G#, MaterialBrief), ... , NextGroup, Replan, Resched, Init} Ω = {ω|ω: <0,1> → X } Q = { q_0, q_1, q_2, q_3, ... , q_{31} } such that q_0=(P_{O1-1},P_{O2-1},P_{O3-1},P_{O4-1},P_{O5-1},P_{O6-1},P_{O7-1}, P_{O8-1},P_{O9-0}), q_1=(P_{O1-1},P_{O2-1},P_{O3-1},P_{O4-1},P_{O5-1},P_{O6-1},P_{O7-1}, P_{O8-1},P_{O9-1}), q_2=(P_{O1-1},P_{O2-1},P_{O3-1},P_{O4-1},P_{O5-1},P_{O6-1},P_{O7-1}, P_{O8-1},P_{O9-2}), ... , and q_{31}=(P_{O1-1},P_{O2-1},P_{O3-1},P_{O4-1},P_{O5-1},P_{O6-1},P_{O7-1}, P_{O8-1},P_{O9-0}) Y = {(O#,OrderBrief),(O#,T#),(T#,GroupCnt-#), (T#,I#,ItemSpec),(O#,T#,I#,G#),(T#,G#, MaterialBrief), ... , NextGroup, Replan, Resched, Init} δ (q)= (δ_{O1} ° proj I_{O1}(q), δ_{O2} ° proj I_{O2}(q), δ_{O3} ° proj I_{O3}(q), ... , δ_{O9} ° proj I_{O9}(q)), q ∈ { q_0, q_1, q_2, q_3, ... , q_{31} } λ = { λ_1, λ_2, λ_3, ... , λ_{31} }

에 의해 상태전이를 발생시키는 상태전이 함수로써 $\delta(q)=(\delta_{O1}$ ° $projI_{O1}(q), \delta_{O2}$ ° $projI_{O2}(q), \delta_{O3}$ ° $projI_{O3}(q)$, ... , δ_{O9} ° $projI_{O9}(q))$와 같다. 즉, 입출력 시스템의 상태전이는 구조적 입출력 시스템의 각 오브젝트에 영향을 주는 Influencer에 의한 상태전이를 통해 이루어진다.

- λ는 λ: Q→Y와 같이 특정 상태에서 출력을 발생시키는 함수로써 $\lambda=\{\lambda_1, \lambda_2, \lambda_3, ..., \lambda_{31}\}$로 구성된다.

4.1.4 DEVS 모델 (DEVS Model)

DEVS 형식론을 기반으로 시간의 흐름에 따라 입력, 상태, 출력, 상태전이 등을 <표 6>처럼 추상화하여 모델링한 시뮬레이션 모델이다 [3,4,5].

DEVS 모델은 $<X,S,Y,\delta_{int},\delta_{ext},\lambda,ta>$와 같은 표현식을 갖는다.

- X는 DEVS 모델의 입력 집합으로써 입출력 시스템의 입력과 동일하다.
- S는 모델의 상태 집합으로써 S={Passive, TGA-1,TGA-2,...} 등으로 구성된다.
- Y는 출력 집합으로써 입력 집합과 동일하다.
- δ_{int}는 내부전이 함수로써 외부입력 없이 상태가 전이된다.
- δ_{ext}는 외부전이 함수로써 외부입력에 의해 다음 상태로 전이된다.
- λ는 λ: Q→Y와 같이 특정 상태에서 출력을 발생시키는 함수로써 $\lambda=\{\lambda_1, \lambda_2, \lambda_3, ..., \lambda_{31}\}$로 구성된다.

<표 6> DEVS 모델의 정의 및 적용 예

정의	M (model) = $<X,S,Y,\delta_{int},\delta_{ext},\lambda,ta>$ where X: set of external (input) event types S: sequential state set Y: output set δ_{int} : S→S, the internal transition function δ_{ext} : Q×X→S, the external transition function ta : S→$R^{+\infty}_0$, the time advance function λ : S→Y, the output function where Q={(s,e)| s∈S, 0≤e≤ta(s)}
적용	M = $<X,S,Y,\delta_{int},\delta_{ext},\lambda,ta>$ where X = {(O#,OrderBrief),(O#,T#),(T#,GroupCnt-#), (T#,I#,ItemSpec),(O#,T#,I#,G#),(T#,G#, MaterialBrief), ... , NextGroup, Replan, Resched, Init}, x∈X S = {Passive,TGA-1,TGA-2,...,Processed}, s∈S Y = {(O#,OrderBrief),(O#,T#),(T#,GroupCnt-#), (T#,I#,ItemSpec),(O#,T#,I#,G#),(T#,G#, MaterialBrief), ... , NextGroup, Replan, Resched, Init}, y∈Y δ_{int}(s) = s δ_{ext}(s e x) = s λ(s) = { λ_1, λ_2, λ_3, ... , λ_9 }

4.2 구조 관계

4.1절에서 살펴본 각 시스템간의 관계를 정의해주는 것이 구조 관계이다. 형식 명세에 의해 인터페이스 시스템을 구조적 입출력 시스템의 요소들로 표현하고, 이를 관계 사상에 의해 입출력 시스템으로 변환한 다음 DEVS 모델링에 의해 DEVS 모델로 재구성한다. 각각의 내용과 적용된 예를 살펴보면 다음과 같다.

4.2.1 형식 명세 (Formal Specification)

인터페이스 시스템을 구조적 입출력 시스템의 요소들로 추상화하여 표현한다. <표 7>을 보면 인터페이스 시스템과 구조적 입출력 시스템의 구조 관계를 알 수 있다. 그리고 <표 8>은 <표 7>에서 기술한 내용을 실제로 적용한 예이다.

<표 8>에서 D는 인터페이스 시스템의 클래스에서 생성한 각 오브젝트(object)이고, Q는 각 클래스에서 수행되는 절차상 구분이 되는 작업 상태에 따라 구분한 상태 집합이다. Var.는 각 오브젝트의 상태변수, δ는 각 상태의 전이를 발생시키는 로컬 상태전이 함수이다.

<표 7> 형식 명세의 정의 및 적용 예

정의	객체지향 프로그래밍에 의해 구성된 인터페이스 시스 템과 구조적 입출력 시스템 $(D,\{Q_\alpha\},\{I_\alpha\},\{\delta_\alpha\})$ 과의 구조 관계를 보면, - 인터페이스 시스템의 각 클래스마다 오브젝트를 생성하여 구조적 입출력 시스템의 D(coordinate)로 정의한다. - 각 클래스는 상태변수에 의해 오브젝트의 상태 집합 Q_α로 표현된다.
적용	D : COrderBriefView × CPlanItemSpecView × ... × CUIApp ⇒ O1 × O2 × ... × O9 Q : COrderBirefView.p1 × CPlanItemSpecView.p2 × ... × CUIApp.p9 ⇒ $P_{O1} \times P_{O2} \times P_{O3} \times ... \times P_{O9}$

<표 8> 인터페이스 시스템과 구조적 입출력 시스템과의 관계

구조적 입출력 시스템				인터페이스 시스템	
D	δ	Var.	Q	클래스	멤버함수
O1	δ_{O1}	P_{O1}	$P_{O1\text{-}1}$	COrderBriefView	OnOK()
			$P_{O1\text{-}2}$		OnInit()
O2	δ_{O2}	P_{O2}	$P_{O2\text{-}1}$	CPlanItemSpecView	OnOK()
			$P_{O2\text{-}2}$		
			$P_{O2\text{-}3}$		
			$P_{O2\text{-}4}$		
			$P_{O2\text{-}5}$		OnInit()
O3	δ_{O3}	P_{O3}	$P_{O3\text{-}1}$	CPlanGenrView	OnGenrPlan()
			$P_{O3\text{-}2}$		
			$P_{O3\text{-}3}$		OnInit()
O4	δ_{O4}	P_{O4}	$P_{O4\text{-}1}$	CSchedGenrView	OnGenrSched()
			$P_{O4\text{-}2}$		
			$P_{O4\text{-}3}$		
			$P_{O4\text{-}4}$		
			$P_{O4\text{-}5}$		
			$P_{O4\text{-}6}$		
			$P_{O4\text{-}7}$		
			$P_{O4\text{-}8}$		OnInit()
O5	δ_{O5}	P_{O5}	$P_{O5\text{-}1}$	CSchedEachProcView	OnOK()
			$P_{O5\text{-}2}$		
O6	δ_{O6}	P_{O6}	$P_{O6\text{-}1}$	CMSUIView	OnOK()
			$P_{O6\text{-}2}$		
O7	δ_{O7}	P_{O7}	$P_{O7\text{-}1}$	CTSUIView	OnOK()
			$P_{O7\text{-}2}$		
O8	δ_{O8}	P_{O8}	$P_{O8\text{-}1}$	CGSUIView	OnOK()
			$P_{O8\text{-}2}$		
O9	δ_{O9}	P_{O9}	$P_{O9\text{-}0}$	CUIApp	OnInitalize()
			$P_{O9\text{-}1}$		InitInstance()
			$P_{O9\text{-}2}$		
			$P_{O9\text{-}3}$		
			$P_{O9\text{-}4}$		
			$P_{O9\text{-}5}$		
			...		
			$P_{O9\text{-}25}$		

4.2.2 관계 사상 (Association Mapping)

구조적 입출력 시스템에서 입출력 시스템으로의 상태전이와 상태전이함수를 <표 9>에서 볼 수 있고, 입력 세그먼트에 의한 상태전이 과정은 <표 10>과 같다.

<표 9> 관계 사상의 정의 및 적용 예

정의	M = <Q, δ>은 다음과 같은 구조를 갖는다. D (coordinates), {Q α | α ∈D}(range sets) {I α | α ∈D, I α ⊆D}(influencers), { δ α | α ∈D, δ α : × β ∈I α Q β → Q α } (local transition functions) 구조적 입출력 시스템과의 관계 정의는 다음과 같다. Q ⊆ × α ∈D Q α, δ = × α ∈D δ α ° projI α restricted to Q
적용	Q: {P01, P02, P03, ... , P09} ⇒ {q0, q1, q2, q3, ... , q31} such that q0=(P01-1,P02-1,P03-1,P04-1,P05-1,P06-1,P07-1,P08-1,P09-0) q1=(P01-1,P02-1,P03-1,P04-1,P05-1,P06-1,P07-1,P08-1,P09-1) q2=(P01-1,P02-1,P03-1,P04-1,P05-1,P06-1,P07-1,P08-1,P09-2),... q31=(P01-1,P02-1,P03-1,P04-1,P05-1,P06-1,P07-1,P08-1,P09-0) δ: δ(q)= (δ 01 ° proj I01(q), δ 02 ° proj I02(q), δ 03 ° proj I03(q), ... , δ 09 ° proj I09(q)), q ∈ { q0, q1, q2, q3, ... , q31 }

<표 9>와 같이 구조적 입출력 시스템의 P_{01} ~ P_{09}의 상태를 입출력 시스템에서 q_{00} ~ q_{031}로 변환된다. 즉, 입출력 시스템의 상태는 구조적 입출력 시스템의 9개 오브젝트의 상태 변수들로 구성되므로, Influencer에 의해 상태전이가 발생하는 오브젝트가 하나라도 존재하면 입출력 시스템에서 각기 다른 상태로 구분된다. 그러므로 <표 10>과 같이 입출력 시스템의 상태가 q_{00} ~ q_{031}로 이루어져 있고, 각 입출력 시스템의 상태는 입력값에 의해 상태전이를 하게 된다. 예를 들어 (O#, OrderBrief)와 같은 입력이 들어오면 입출력 시스템의 상태는 'q_0'에서 'q_1'으로 전이되고, 그때의 구조적 입출력 시스템의 상태를 보면 O1부터 O8까지의 오브젝트는 초기상태 그대로이고, O9 오브젝트의 상태가 'P_{09-0}'에서 'P_{09-1}'로 전이되었음을 알 수 있다. <표 10>에서 굵은 글씨체로 표시된 것은 입출력 시스템의 상태를 이루는 요소가 되는 구조적 입출력 시스템의 상태가 전이되었음을 나타낸다. 위와 같이 Influencer에 의한 상태전이의 모든 경우에 대해서 수동으로 파악하여 입출력 시스템의 상태를 구성해야 하는 어려움이 있지만, 추후 소프트웨어 자동 구축 또는 변경관리를 하기 위한 자동 시스템을 구현하기 위한 이론적인 토대를 마련하기 위한 과정이라 할 수 있다.

<표 10> 구조적 입출력 시스템과 입출력 시스템과의 관계

X	Q × δ → Q
(O#,OrderBrief)	q_1=(P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},P_{09-1})
(O#,T#)	q_2=(P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-2}**)
(T#, GroupCnt-#)	q_3=(P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-3}**)
(T#,Replan-on/off)	q_4=(**P_{01-2}**,P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-3}**)
Init	q_5=(**P_{01-1}**,P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-25}**)
(T#,I#,ItemSpec)	q_6=(P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-5}**)
(O#,T#,I#,G#)	q_7=(P_{01-1},**P_{02-2}**,P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-6}**)
(T#,G#,MaterialBrief)	q_8=(P_{01-1},**P_{02-3}**,P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-7}**)
(T#,G#,ManufactSpec)	q_9=(P_{01-1},**P_{02-4}**,P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-8}**)
(T#,G#,StandardItem)	q_{10}=(P_{01-1},**P_{02-5}**,P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-9}**)
NextGroup/Replan/Init	q_{11}=(P_{01-1},**P_{02-1}**,P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-25}**)
(T#,G#,PlanCond)	q_{12}=(P_{01-1},P_{02-1},**P_{03-2}**,P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-10}**)
(T#,G#,CutPlan)	q_{13}=(P_{01-1},P_{02-1},P_{03-2},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-11}**)
(O#,T#,I#,G#,P#)	q_{14}=(P_{01-1},P_{02-1},**P_{03-3}**,P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-12}**)
NextGroup/Init	q_{15}=(P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-25}**)
(T#,G#,P#,MachineBrief)	q_{16}=(P_{01-1},P_{02-1},P_{03-1},**P_{04-2}**,P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-13}**)
(T#,G#,P#,TeamBrief)	q_{17}=(P_{01-1},P_{02-1},P_{03-1},**P_{04-3}**,P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-14}**)
(T#,G#,P#,GalvBrief)	q_{18}=(P_{01-1},P_{02-1},P_{03-1},**P_{04-4}**,P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-15}**)
(T#,G#,P#,WorkDay)	q_{19}=(P_{01-1},P_{02-1},P_{03-1},**P_{04-5}**,P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-16}**)
:	:
NextGroup/Init	q_{31}=(P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},**P_{09-25}**)
()	q_0=(P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1},P_{09-0})

4.2.3 DEVS 모델링 (DEVS Modeling)

입출력 시스템에서 <표 11>과 같이 DEVS 형식론을 적용하여 DEVS 모델로 모델링하였다. 입출력 시스템과 DEVS 모델과의 구조 관계는 <표 12>과 같다.

<표 11>에서 입출력 시스템의 q_{00} ~ q_{031}로 구성된 상태는 DEVS 모델에서 PASSIVE, TGA-1, TGA-2 등으로 변환된다. <표 12>와 같이 외부 입력(X)에 의해 상태전이 함수(δ)는

다음 상태(S)로 상태 전이시키고, 출력 함수에 의해 출력값(Y)을 생성한다. 예를 들어 DEVS 모델의 초기상태인 'PASSIVE'에서 ((O-,#), OrderBrief)와 같은 입력이 들어오면 상태전이 함수에 의해 'TGA-1'로 상태전이 되고, 이때의 출력은 (TGA-1, ((O-,#),OrderBrief)이다. 이 출력을 통해 입력값이나 모델의 상태를 파악할 수 있다. 또한 이때 대응되는 입출력 시스템의 상태는 'q_1'임을 알 수 있다. 모델의 상태가 'GGA-5'일 때 NextGroup이라는 외부 입력이 들어오면 상태전이 함수에 의해 'TGA-5'로 상태전이되고, 출력은 (TGA-5, NextGroup)이다. Init이라는 외부 입력이 들어오면 'GGA-6'으로 상태전이되고, 출력은 (GGA-6, Init)이다. 이때 대응되는 입출력 시스템의 상태는 'q_{11}'이다.

<표 11> DEVS 모델링의 정의 및 적용 예

정의	M (model) = <X_M, S_M, Y_M, δ_{int}, δ_{ext}, λ_M, ta> where X_M = X' S_M = S' Y_M = Y' δ_{int} : $Q_M \times \oint$ -> Q_M δ_{ext} : $Q_M \times \Omega'$ -> Q_M, where Q_M ={(s,e)|s∈S_M, 0≤e≤ta(s)} ta : S->Real λ_M : λ'
적용	S : { q_0, q_1, q_2, q_3, ... , q_{31} } ⇒ {PASSIVE,TGA-1,TGA-2,TGA-3, ... ,PROC} δ(q) ⇒ δ_{ext}

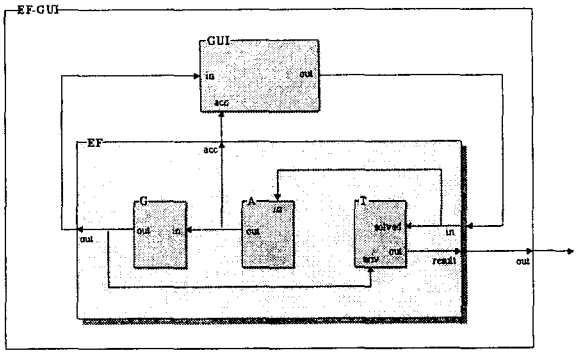
<표 12> 입출력 시스템과 DEVS 모델과의 관계

DEVS 모델				입출력 시스템
S	X	δ	Y	Q
Passive	((O-,#),OrderBrief)	TGA-1	(TGA-1,(O-,#),OrderBrief)	q_1
TGA-1	((O-,#),(T-,#))	TGA-2	(TGA-2,(O-,#),(T-#))	q_2
TGA-2	((T-,#),(GroupCnt,#))	TGA-3	(TGA-3,(T-,#),(GroupCnt,#))	q_3
TGA-3	((T-,#),(Replan,#))	TGA-4	(TGA-4,(T-,#),(Replan,#))	q_4
TGA-4	Init	TGA-5	(TGA-5,Init)	q_5
TGA-5	(((T-,#),(I-,#),ItemSpec)	GGA-1	(GGA-1,((T-,#),(I-,#),ItemSpec)	q_6
GGA-1	((O-,#),(T-,#),(I-,#),(G-,#))	GGA-2	(GGA-2,(O-,#),(T-,#),(I-,#),(G-,#))	q_7
GGA-2	(((T-,#),(G-,#)),MaterialBrief)	GGA-3	(GGA-3,((T-,#),(G-,#)),MaterialBrief)	q_8
GGA-3	(((T-,#),(G-,#)),ManufactSpec)	GGA-4	(GGA-4,((T-,#),(G-,#)),ManufactSpec)	q_9
GGA-4	(((T-,#),(G-,#)),StandardItem)	GGA-5	(GGA-5,((T-,#),(G-,#)),StandardItem)	q_{10}
GGA-5	NextGroup	TGA-5	(GGA-5,NextGroup)	q_{11}
	Init	GGA-6	(GGA-6,Init)	
GGA-6	(((T-,#),(G-,#)),PlanCond)	TRA-1	(TRA-1,(((T-,#),(G-,#)),PlanCond)	q_{12}
TRA-1	(((T-,#),(G-,#)),CutPlan)	TRA-2	(TRA-2,(((T-,#),(G-,#)),CutPlan)	q_{13}
TRA-2	((O-,#),(T-,#),(I-,#),(G-,#),(P-,#))	TRA-3	(TRA-3,(O-,#),(T-,#),(I-,#),(G-,#),(P-,#))	q_{14}
TRA-3	NextGroup	GGA-6	(GGA-6,NextGroup)	q_{15}
	Replan	GGA-6	(GGA-6,Replan)	
	(((T-,#),(G-,#),(P-,#)),MachineBrief)	TSA-1	(TSA-1,(((T-,#),(G-,#),(P-,#)),MachineBrief)	
TRA-4	(((T-,#),(G-,#),(P-,#)),MachineBrief)	TSA-1	(TSA-1,(((T-,#),(G-,#),(P-,#)),MachineBrief)	q_{16}
	...		...	...
GPA-1	NextGroup	TPA-2	(TPA-2,NextGroup)	q_{31}
	Replan	GPA-2	(GPA-2,Init)	
GPA-2	()	PROC	Processed	q_0

5. 시뮬레이션 결과

이 장에서는 4장에서 기술된 시스템 명세의 계층 구조를 통해 인터페이스 시스템을 DEVS 모델로 모델링하고 시뮬레이션 함으로써 인터페이스 시스템의 중요한 동적 특성을 소프트웨어 설계시 또는 설계 변경 후에 미리 파악하고자 한다.

GUI 모델은 4장에서 기술된 시스템 명세의 계층 구조에 의해 DEVS 모델로 추상화된 인터페이스 시스템이다. EF(Experimental Frame)는 GUI 모델과 연결된 coupled 모델로서 입력 세그먼트를 발생시키고, 시뮬레이션의 흐름을 제어하거나, 시뮬레이션 된 내용이나 통계적인 결과를 생성하는 등의 역할을 하고, 3개의 컴포넌트 모델(GENR, ACCPTR, TRANSD)로 구성된다[3,4]. EF-GUI의 구조는 <그림 3>과 같다. EF-GUI 모델을 DEVS-Scheme을 이용해서 시뮬레이션 결과를 통하여 유용성을 검증하도록 하겠다.



<그림 3> EF-GUI 구조

다음은 DEVS 모델의 상태를 구분하는데 근거가 되는 사용자 인터페이스의 화면에 따른 처리 절차이다.

● 화면(Action)별 처리절차

1. TGA(Task Generation Action) : 수주정보가 입력되고, 태스크 번호가 생성된 상태
2. GGA(Group Generation Action) : 태스크당 품목정보가 입력되고, 그룹번호가 생성된 상태
3. TRA(Task Re-organization(planning) Action) : 절단계획에 필요한 조건이 입력되고, 절단계획이 (재)생성된 상태
4. TSA(Task Scheduling Action) : 스케줄에 필요한 조건이 입력되고, 스케줄이 (재)생성된 상태
5. MPA(Machine Processing Action) : 머신공정의 일정이 처리된 상태
6. TPA(Team Processing Action) : 가공공정의 일정이 처리된 상태
7. GPA(Galv Processing Action) : 도금공정의 일정이 처리된 상태

다음은 4장에서 살펴본 구조 관계를 통해 각 시스템간의 상호 의존성을 추적한 예이다. 각 시스템간의 상호 의존성 추적을 통해 구조 관계를 통한 시스템 변환 과정의 유용성을 살펴 볼 수 있다. 시스템간의 구조 관계를 자동 변환하는 기능은 추후 해결해야 할 과제이다.

● 화면별 처리절차에 따른 상호 의존성 추적 예

1. TGA (Task Generation Action) :
   ① 수주번호가 생성되고, 수주의 정보가 입력된 상태
   ② 태스크 번호가 생성된 상태
   ③ 해당 태스크의 그룹 개수가 정해진 상태
   ④ 절단계획 재생성 여부가 결정된 상태
   ⑤ TGA가 초기화된 상태

<표 13> TGA의 상호 의존성 추적

DEVS 모델	입출력 시스템	구조적 입출력 시스템	인터페이스 시스템
TGA-1	q_1	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}1}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}1}$	CUIApp.InitInstance()
TGA-2	q_2	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}1}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}2}$	CUIApp.InitInstance()
TGA-3	q_3	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}1}$,$P_{06\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\ 1}$,$P_{09\text{-}3}$	CUIApp.InitInstance()
TGA-4	q_4	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}1}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}4}$	COrderBriefView.OnOK() CUIApp.InitInstance()
TGA-5	q_5	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}1}$,$P_{05\ 1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}25}$	COrderBriefView.OnInit() CUIApp.OnInitialize()

2. GGA (Group Generation Action) :
   ① 품목번호가 생성되고, 품목의 명세가 입력된 상태
   ② 그룹번호가 생성된 상태
   ③ 자재개요가 입력된 상태
   ④ 제작사양가 입력된 상태
   ⑤ 규격품목가 입력된 상태
   ⑥ 그룹개수가 1개 이상일 경우 그룹 개수만큼 ①~⑤를 반복 수행하기 위해 ①로 되돌아간 상태
   ⑦ GGA가 초기화된 상태

<표 14> GGA의 상호 의존성 추적

DEVS 모델	입출력 시스템	구조적 입출력 시스템	인터페이스 시스템
GGA-1	q_6	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}1}$,$P_{05\ 1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}5}$	CUIApp.InitInstance()
GGA-2	q_7	$P_{01\text{-}1}$,$P_{02\text{-}2}$,$P_{03\ 1}$,$P_{04\text{-}1}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}6}$	CPlanItemSpecView.OnOK() CUIApp.InitInstance()
GGA-3	q_8	$P_{01\text{-}1}$,$P_{02\text{-}3}$,$P_{03\text{-}1}$,$P_{04\text{-}1}$,$P_{05\ 1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}7}$	CPlanItemSpecView.OnOK() CUIApp.InitInstance()
GGA-4	q_9	$P_{01\text{-}1}$,$P_{02\text{-}4}$,$P_{03\text{-}1}$,$P_{04\text{-}1}$,$P_{05\ 1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}8}$	CPlanItemSpecView.OnOK() CUIApp.InitInstance()
GGA-5	q_{10}	$P_{01\text{-}1}$,$P_{02\text{-}5}$,$P_{03\text{-}1}$,$P_{04\ 1}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}9}$	CPlanItemSpecView.OnOK() CUIApp.InitInstance()
GGA-6	q_{11}	$P_{01\text{-}1}$,$P_{02\ 1}$,$P_{03\text{-}1}$,$P_{04\ 1}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}25}$	CPlanItemSpecView.OnInit() CPlanItemSpecView.OnRefresh() CUIApp.OnInitialize()

3. TRA (Task Re-organization(Planning) Action):

① 그룹별 계획 조건이 입력된 상태
② 해당 그룹의 절단계획 생성 명령이 떨어진 상태
③ 계획번호가 생성된 상태
④ 그룹개수가 1개 이상일 경우 그룹 개수만큼 ①~③를 반복 수행하기 위해 ①로 되돌아간 상태
⑤ 절단계획을 재생성할 경우 ①~④를 반복 수행하기 위해 ①로 되돌아간 상태
⑥ 초기화

<표 15> TRA의 상호 의존성 추적

DEVS 모델	입출력 시스템	구조적 입출력 시스템	인터페이스 시스템
TRA-1	q_{12}	P_{01-1},P_{02-1},P_{03-2},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-10}	CPlanGenrView.OnGenPlan() CUIApp.InitInstance()
TRA-2	q_{13}	P_{01-1},P_{02-1},P_{03-2},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-11}	CUIApp.InitInstance()
TRA-3	q_{14}	P_{01-1},P_{02-1},P_{03-3},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-12}	CPlanGenrView.OnGenPlan() CUIApp.InitInstance()
TRA-4	q_{15}	P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-25}	CPlanGenrView.OnOK() CUIApp.OnInitialize()

시뮬레이션 결과를 아래와 같이 크게 두 가지 경우로 나누어 볼 수 있다. 우선 DEVS 모델에 을 DEVS 시뮬레이션 환경에서 제공하는 시뮬레이터를 통해 시뮬레이션 했을 때 정상적으로 종료될 경우 수행되는 과정을 살펴보고, 비정상적으로 중단될 경우 DEVS 모델의 입력과 출력을 통해 중단된 위치에서의 모델의 상태와 계층적 시스템간의 상호 의존성을 통해 인터페이스 시스템에 미치는 영향 등을 추적하는 과정을 예를 통해 살펴보겠다.

● 시뮬레이션이 정상 종료된 경우

예를 들어, <표 16>과 같이 DEVS 모델에 외부 입력에 의한 출력을 통해 ③ 과정은 DEVS 모델의 'GGA-3' 상태임을 알 수 있고, <표 17>를 통해 모델의 상태 'GGA-3'은 입출력 시스템의 'q_8', 구조적 입출력 시스템의 9개 컴포넌트의 상태는 각각 'P_{01-1},P_{02-3},P_{03-1},P_{04-1}, P_{05-1},P_{06-1},P_{07-1},P_{08-1}, P_{09-7}'이고, 이때의 인터페이스 시스템의 클래스는 CPlanItemSpecView와 CUIApp임을 알 수 있다.

<표 16> GGA의 정상적인 시뮬레이션 결과

	입력	출력
①	(((T- 17) (I- 53)) ITEMSPEC)	(GGA-1 ((T- 17) (I- 53)) ITEMSPEC)
②	((O- 7) (T- 17) (I- 53) (G- 42))	(GGA-2 (O- 7) (T- 17) (I- 53) (G- 42))
③	(((T- 17) (G- 42) MATERIALBRIEF)	(GGA-3 ((T- 17) (G- 42)) MATERIALBRIEF)
④	(((T- 17) (G- 42)) MANUFACTSPEC)	(GGA-4 ((T- 17) (G- 42)) MANUFACTSPEC)
⑤	(((T- 17) (G- 42)) STANDARDITEM)	(GGA-5 ((T- 17) (G- 42)) STANDARDITEM)
⑥	NEXTGROUP	(TGA-5 . NEXTGROUP)
①	(((T- 17) (I- 71)) ITEMSPEC)	(GGA-1 ((T- 17) (I- 71)) ITEMSPEC)
②	((O- 7) (T- 17) (I- 53) (G- 42) (I- 71) (G- 60))	(GGA-2 (O- 7) (T- 17) (I- 53) (G- 42) (I- 71) (G- 60))
③	(((T- 17) (G- 60)) MATERIALBRIEF)	(GGA-3 ((T- 17) (G- 60)) MATERIALBRIEF)
④	(((T- 17) (G- 60)) MANUFACTSPEC)	(GGA-4 ((T- 17) (G- 60)) MANUFACTSPEC)
⑤	(((T- 17) (G- 60)) STANDARDITEM)	(GGA-5 ((T- 17) (G- 60)) STANDARDITEM)
⑦	INIT	(GGA-6 . INIT)

● 시뮬레이션이 비정상 종료된 경우

【예 1】 DEVS 모델에서 특정 상태전이(State transition)가 변경되었을 때 인터페이스 시스템에 미치는 영향 등을 살펴보겠다.

<표 17>를 보면 ⑦ 과정에서 시뮬레이션이 비정상 종료되었다. DEVS 모델에서 중단된 부분 ⑦에서 각 시스템간의 상호 의존성을 추적해 보면 DEVS 모델은 'GGA-6', 입출력 시스템은 'q_{11}', 구조적 입출력 시스템의 각 컴포넌트의 상태는'P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1},P_{06-1},P_{07-1},P_{08-1}, P_{09-25}'이다. 그러므로 인터페이스 시스템에서 시뮬레이션이 중단된 부분에 해당되는 클래스는 'CPlanItemSpecView'와 'CUIApp'이다.

위와 같은 클래스에서 오류가 발생한 이유는 'GGA-5'에서 'TGA-5'로의 State transition이 'TGA-3'으로의 State transition으로 변경됨으로써 영향을 받는 'CPlanItemSpecView'의 'OnRefresh()'와 'CUIApp'의 'InitInstance()'의

변경이 제대로 이루어지지 않았기 때문이다. 이러한 시스템간의 상호 연관성 추적과 변경 관리는 <표 18>과 <그림 4>, <그림 5>를 통해 알 수 있다.

GGA (Group Generation Action) :

① 품목번호가 생성되고, 품목의 명세가 입력된 상태
② 그룹번호가 생성된 상태
③ 자재개요가 입력된 상태
④ 제작사양이 입력된 상태
⑤ 규격품목이 입력된 상태
⑥ 그룹개수가 1개 이상일 경우 그룹 개수만큼 ①~⑤를 반복 수행하기 위해 ③로 되돌아간 상태 ⇨ *변경!!*
⑦ GGA가 초기화된 상태

<표 17> State transition 변경 후의 시뮬레이션 결과

	입력	출력
①	(((T- 58) (I- 11)) ITEMSPEC)	(GGA-1 ((T- 58) (I- 11)) ITEMSPEC)
②	((O- 32) (T- 58) (I- 11) (G- 27))	(GGA-2 (O- 32) (T- 58) (I- 11) (G- 27))
③	(((T- 58) (G- 27) MATERIALBRIEF)	(GGA-3 ((T- 58) (G- 27)) MATERIALBRIEF)
④	(((T- 58) (G- 27)) MANUFACTSPEC)	(GGA-4 ((T- 58) (G- 27)) MANUFACTSPEC)
⑤	(((T- 58) (G- 27)) STANDARDITEM)	(GGA-5 ((T- 58) (G- 27)) STANDARDITEM)
⑥	NEXTGROUP	(TGA-5 . NEXTGROUP)
③	(((T- 58) (G- 82))MATERIALBRIEF)	(GGA-3 ((T- 58) (G- 82)) MATERIALBRIEF)
④	(((T- 58) (G- 82)) MANUFACTSPEC)	(GGA-4 ((T- 58) (G- 82)) MANUFACTSPEC)
⑤	(((T- 58) (G- 82)) STANDARDITEM)	(GGA-5 ((T- 58) (G- 82)) STANDARDITEM)
⑦	INIT	

【예 2】 DEVS 모델에서 특정 상태(State)를 삭제했을 때 인터페이스 시스템에 미치는 영향 등을 살펴보겠다.

<표 19>를 보면 ⑫ 과정에서 시뮬레이션이 비정상 종료되었다. DEVS 모델에서 중단된 부분 ⑫에서 각 시스템간의 상호 의존성을 추적해보면 DEVS 모델은 'TSA-10', 입출력 시스템은 'q_{25}', 구조적 입출력 시스템의 각 컴포넌트의 상태는 'P_{01-1},P_{02-1},P_{03-1},P_{04-8},P_{05-1},P_{06-1},P_{07-1},P_{08-1},P_{09-25}' 이다. 그러므로 인터페이스 시스템에서 시뮬레이션이 중단된 부분에 해당되는 클래스는 CSched GenrView와 CSchedEachProcView이다.

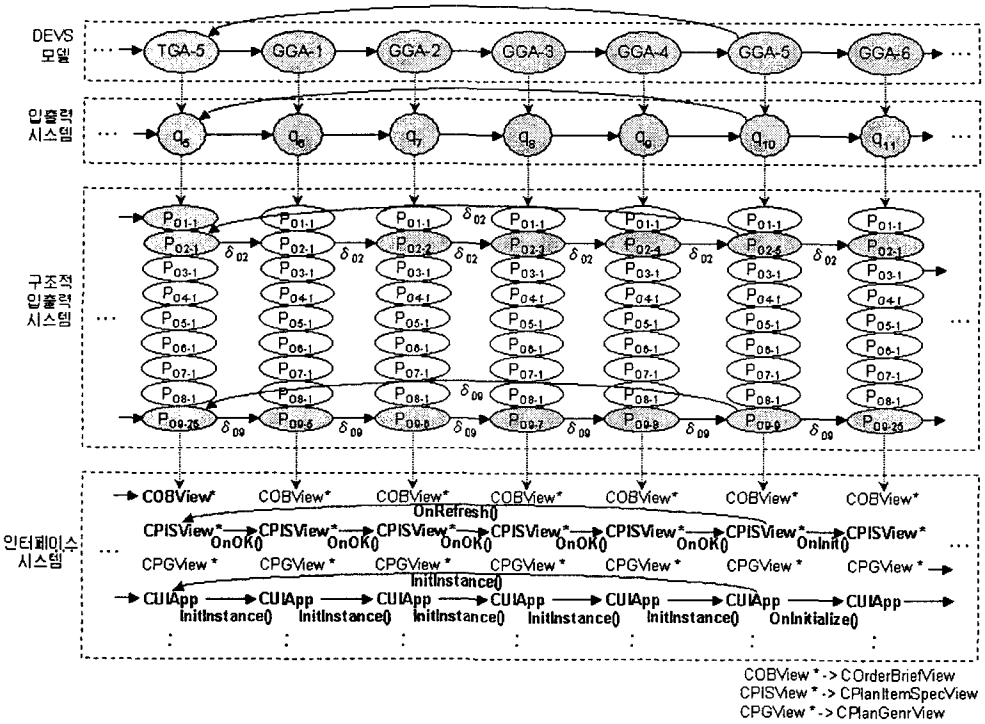
<표 18> GGA의 상호 의존성 추적

DEVS 모델	입출력 시스템	구조적 입출력 시스템	인터페이스 시스템
GGA-1	q_5	P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-5}	CUIApp.InitInstance()
GGA-2	q_7	P_{01-1},P_{02-2},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-6}	CPlanItemSpecView.OnOK() CUIApp.InitInstance()
GGA-3	q_8	P_{01-1},P_{02-3},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-7}	CPlanItemSpecView.OnOK() CUIApp.InitInstance()
GGA-4	q_9	P_{01-1},P_{02-4},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-8}	CPlanItemSpecView.OnOK() CUIApp.InitInstance()
GGA-5	q_{10}	P_{01-1},P_{02-5},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-9}	CPlanItemSpecView.OnOK() CUIApp.InitInstance()
GGA-6	q_{11}	P_{01-1},P_{02-1},P_{03-1},P_{04-1},P_{05-1}, P_{06-1},P_{07-1},P_{08-1},P_{09-25}	CPlanItemSpecView.OnInit() CPlanItemSpecView.OnRefresh() CUIApp.OnInitialize()

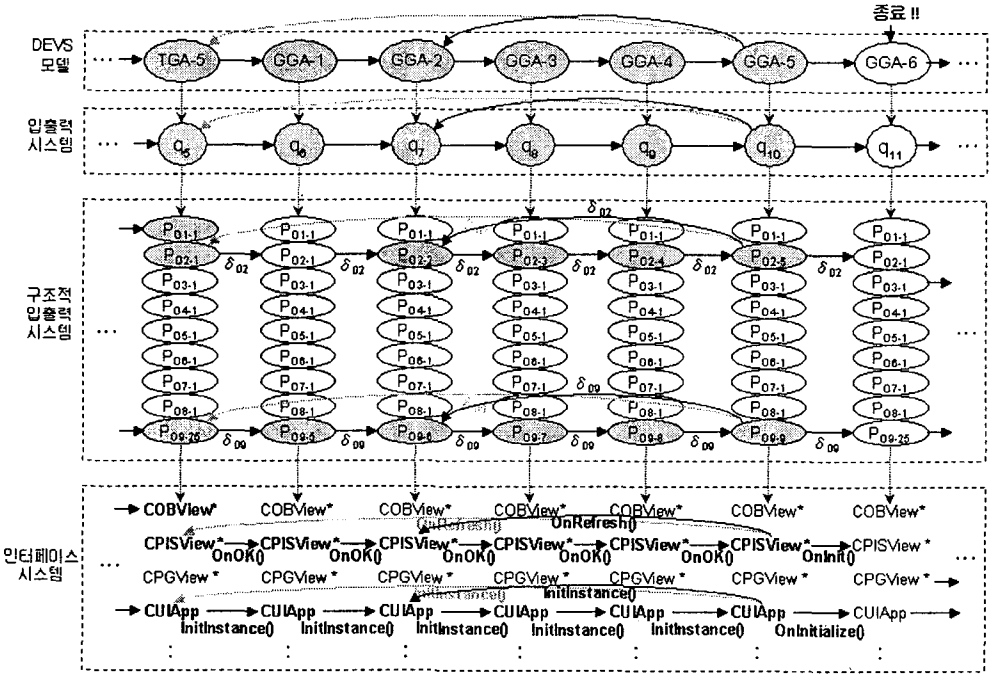
위와 같은 클래스에서 오류가 발생한 이유는 DEVS 모델에서 State 'TSA-8' 삭제를 함으로써 영향을 받는 CSchedGenrView의 OnSched()와 CUIApp의 InitInstance()의 변경이 제대로 이루어지지 않았기 때문이다. 이러한 시스템간의 상호 연관성 추적과 변경 관리는 <표 20>와 <그림 6>, <그림 7>을 통해 알 수 있다.

TSA (Task Schduling Action):

① 그룹별 머신 개요가 입력된 상태
② 가공팀 개요가 입력된 상태
③ 도금팀 개요가 입력된 상태
④ 작업일이 입력된 상태
⑤ 작업시간이 입력된 상태
⑥ 스케쥴 재생성 여부가 결정된 상태
⑦ 공정별 스케쥴이 생성된 상태
⑧ 작업지시 명령이 떨어진 상태 ⇨ *삭제!!*
⑨ 머신일정번호, 가공팀일정번호, 도금팀일정번호가 생성된 상태
⑩ 그룹개수가 1개 이상일 경우 그룹 개수만큼 ①~⑨를 반복수행하기 위해 ①로 되돌아간 상태
⑪ 스케쥴을 재생성할 경우 ①~⑩를 반복 수행하기 위해 ①로 되돌아간 상태
⑫ TSA가 초기화된 상태



<그림 4> State transition 변경 전의 시뮬레이션 결과



<그림 5> State transition 변경 후의 시뮬레이션 결과

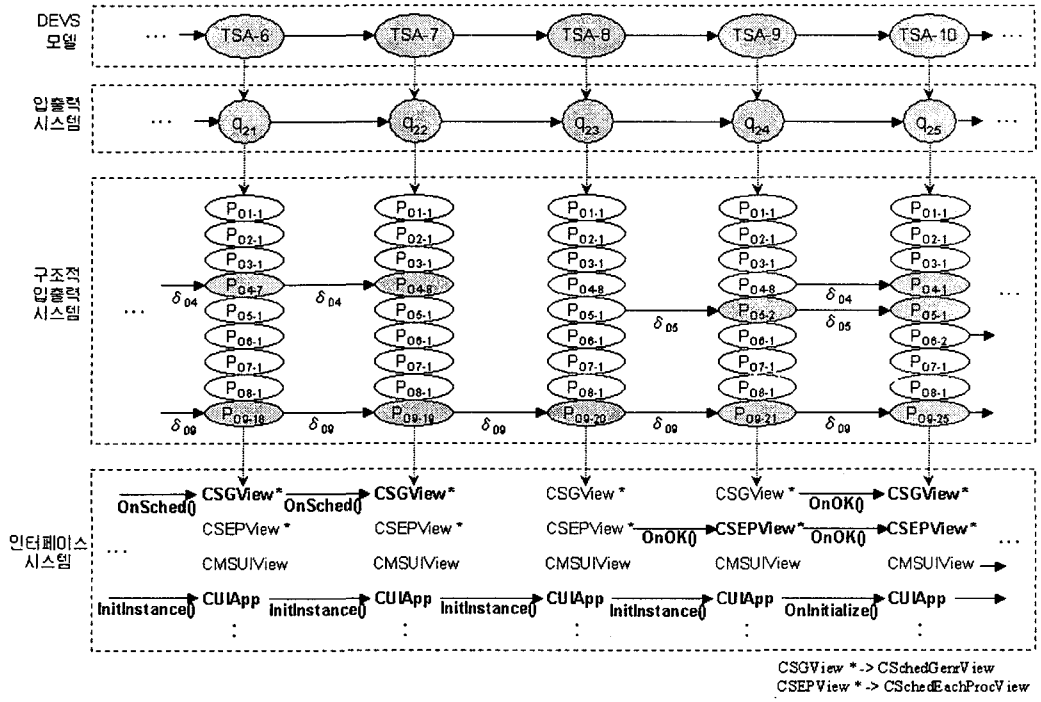
<표 19> State 삭제 후의 시뮬레이션 결과

	입력	출력
①	(((T- 21) (G- 69) (P- 73)) MACHINEBRIEF)	(TSA-1 ((T- 21) (G- 69) (P- 73)) MACHINEBRIEF)
②	(((T- 21) (G- 69) (P- 73)) TEAMBRIEF)	(TSA-2 ((T- 21) (G- 69) (P- 73)) TEAMBRIEF)
③	(((T- 21) (G- 69) (P- 73)) GALVBRIEF)	(TSA-3 ((T- 21) (G- 69) (P- 73)) GALVBRIEF)
④	(((T- 21) (G- 69) (P- 73)) WORKDAY)	(TSA-4 ((T- 21) (G- 69) (P- 73)) WORKDAY)
⑤	(((T- 21) (G- 69) (P- 73)) WORKTIME)	(TSA-5 ((T- 21) (G- 69) (P- 73)) WORKTIME)
⑥	(((T- 21) (G- 69) (P- 73)) (RESCHED 0))	(TSA-6 ((T- 21) (G- 69) (P- 73)) (RESCHED 0))
⑦	(((T- 21) (G- 69) (P- 73)) SCHEDULE)	(TSA-7 ((T- 21) (G- 69) (P- 73)) SCHEDULE)
⑨	((O- 32) (T- 21) (I- 52) (G- 69) (I- 60) (G- 13) (P- 27) (MS- 4) (TS- 91) (GS- 56))	(TSA-9 (O- 32) (T- 21) (I- 52) (G- 69) (I- 60) (G- 13) (P- 27) (MS- 4) (TS- 91) (GS- 56))
⑩	NEXTGROUP	(TRA-4 . NEXTGROUP)
①	(((T- 21) (G- 30) (P- 84)) MACHINEBRIEF)	(TSA-1 ((T- 21) (G- 30) (P- 84)) MACHINEBRIEF)
②	(((T- 21) (G- 30) (P- 84)) TEAMBRIEF)	(TSA-2 ((T- 21) (G- 30) (P- 84)) TEAMBRIEF)
③	(((T- 21) (G- 30) (P- 84)) GALVBRIEF)	(TSA-3 ((T- 21) (G- 30) (P- 84)) GALVBRIEF)
④	(((T- 21) (G- 30) (P- 84)) WORKDAY)	(TSA-4 ((T- 21) (G- 30) (P- 84)) WORKDAY)
⑤	(((T- 21) (G- 30) (P- 84)) WORKTIME)	(TSA-5 ((T- 21) (G- 30) (P- 84)) WORKTIME)
⑥	(((T- 21) (G- 30) (P- 84)) (RESCHED 0))	(TSA-6 ((T- 21) (G- 30) (P- 84)) (RESCHED 0))

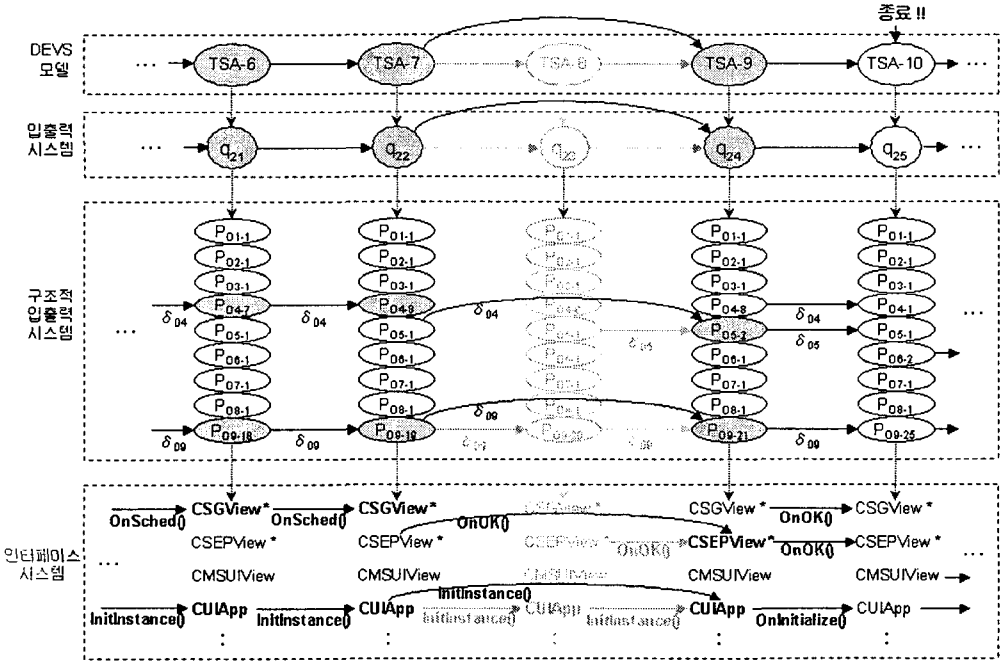
<표 20> TSA의 상호 의존성 추적

DEVS 모델	입출력 시스템	구조적 입출력 시스템	인터페이스 시스템
TSA-1	q_{16}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}2}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}13}$	CSchedGenrView.OnSched() CUIApp.InitInstance()
TSA-2	q_{17}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}3}$,$P_{05\text{-}1}$,$P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}14}$	CSchedGenrView.OnSched() CUIApp.InitInstance()
TSA-3	q_{18}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}4}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}15}$	CSchedGenrView.OnSched() CUIApp.InitInstance()
TSA-4	q_{19}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}5}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}16}$	CSchedGenrView.OnSched() CUIApp.InitInstance()
TSA-5	q_{20}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}6}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}17}$	CSchedGenrView.OnSched() CUIApp.InitInstance()
TSA-6	q_{21}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}7}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}18}$	CSchedGenrView.OnSched() CUIApp.InitInstance()
TSA-7	q_{22}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}8}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}19}$	CSchedGenrView.OnSched() CUIApp.InitInstance()
TSA-8	q_{23}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}8}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}20}$	CSchedGenrView.OnSched() CUIApp.InitInstance()
TSA-9	q_{24}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}8}$,$P_{05\text{-}2}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}21}$	CSchedEachProcView.OnOK() CUIApp.InitInstance()
TSA-10	q_{25}	$P_{01\text{-}1}$,$P_{02\text{-}1}$,$P_{03\text{-}1}$,$P_{04\text{-}8}$,$P_{05\text{-}1}$, $P_{06\text{-}1}$,$P_{07\text{-}1}$,$P_{08\text{-}1}$,$P_{09\text{-}25}$	CSchedGenrView.OnSched() CSchedEachProcView.OnOK() CUIApp.OnInitialize()

위의 두가지 예를 통해 DEVS 모델에서 기존의 상태가 삭제되거나 수정되는 등의 변경이 각 계층적 시스템의 상호 의존성 추적을 통해 인터페이스 시스템에 어떤 영향을 미치는지를 파악할 수 있다. 즉, 복잡하고 규모가 큰 인터페이스 시스템을 직접 변경하지 않고, 추상화된 DEVS 모델로 간단히 시뮬레이션 해봄으로써 변경해야 할 인터페이스 시스템의 해당 위치를 추적하거나, 혹은 변경 후에 미치는 영향 등을 쉽게 파악할 수 있다.



<그림 6> State 삭제 전의 시뮬레이션 결과



<그림 7> State 삭제 후의 시뮬레이션 결과

6. 결 론

인터페이스 시스템은 빈번한 기능 추가 또는 변경에 의한 수정 때문에 자주 프로토타입으로 만들어지고 반복적으로 수정되어야 한다. 그러므로 소프트웨어의 규모가 커지고 복잡해질수록 수정하는 일이 어려워진다[6,9,13]. 그러나 비교적 수행절차에 필요한 알고리즘이 단순하고 처리 과정이 복잡하지 않으므로 사상 이론을 적용하여 자동 시스템을 구현하는 것이 용이하다는 특성을 이용하여 본 논문에서는 시스템 형식론을 적용해서 인터페이스 시스템을 DEVS 모델로 재구성하여 DEVS 시뮬레이션 환경에서 제공하는 시뮬레이터를 통하여 대상 시스템의 중요한 동적 특성을 소프트웨어 설계 시 또는 설계 변경 후 미리 파악할 수 있도록 하였다. 이를 통해 소프트웨어 시스템을 개발할 때마다 반복적으로 되풀이되는 수정에 의한 추적의 어려움을 덜어줄 뿐만 아니라 전체 시스템의 구조나 흐름을 파악하여 변경된 부분을 추적할 수 있도록 하였다.

본 논문에서 지금까지는 수동으로 구조 관계에 의해 각 계층적 시스템으로 변환하고 추상화한 DEVS 모델을 시뮬레이션 한 다음, 그 모델의 결과를 통해 역시 수동으로 상호 의존성이나 변경에 의한 영향 등을 파악하였으나, 추후 이러한 이론적 토대를 바탕으로 자동 추적 기능을 갖는 시스템을 구현하여 소프트웨어 변경 관리를 하고, 나아가서는 함축적이고 추상화된 단순한 DEVS 모델을 모델링한 후 자동적으로 입출력 시스템, 구조적 입출력 시스템을 거쳐 일반 소프트웨어를 구현하는 소프트웨어 구축기를 설계 및 구현이 이루어져야 할 것이다.

그러므로 본 연구에서 시간 개념을 배제한 인터페이스 시스템이라는 특정 도메인을 대상으로 설계 방법론을 제시하였으나, 향후 시간 의존적인 인터페스 시스템뿐만 아니라 확장된 일반 소프트웨어에도 적용 가능한 체계적인 소프트웨어 설계 방법론을 정립하기 위한 연구가 이루어지고 있다

참고 문헌

[1] Bernard.P.Zeigler, Theory of Modelling and Simulation, John Wiley, NY, USA, 1976

[2] L.Padulo and M.A.Arvbib, System Theory, Suders, Philadelpia, Pa, 1974

[3] Bernard.P.Zeigler, Multifacetted Modelling and Discrete Event Simulation Orlando, FL, USA: Academic, 1984

[4] Bernard.P.Zeigler, Object-Oriented Simulation with Hierarchical, Modular Models, San Diego, CA, USA: Academic Press, 1990

[5] 조대호, 이칠기, "계층의 구조를 갖는 시뮬레이션 모델에 있어서 단계적 접근을 위한 모델 연결 방법론과 그 적용 예", 한국시뮬레이션학회 논문지, Vol 5, No 2, 1996

[6] 김상근, 객체지향 기술에 의한 사용자 인터페이스 구축기 개발, 중앙대학교 컴퓨터공학과 소프트웨어 엔지니어링 석사학위 청구논문, 1995.12

[7] 김덕현, 박성주, "확장된 객체지향 데이터 모델을 이용한 소프트웨어 변경 관리 시스템", 정보과학회 논문지, Vol 22, No 2, pp.249-260, 1995

[8] 허계범, 최영근, 객체지향 소프트웨어 공학, 한국실리콘, 1995

[9] 우치수외, 사용자 인터페이스, 영지문화사, 1994

[10] 이경환외, 소프트웨어공학, 청문각, 1993

[11] 이경환외, 소프트웨어 재이용 시스템 개발(II), 연구보고서, 과학기술처, 1993

[12] G. Booch, Object-Oriented Design, Bengamin/Cummings Publishing, 1989

[13] R. Keller, M. Cameron, R. Taylor, D. Troup, "User Interface Development and Software Environments: The Chiron-1 System", Proceedings of the 12th ICSE,

pp. 208-218, 1991

[14] A. Marcus, Graphic Design for Electronic Documents and User Interfaces, Addison-Wesley Publishing Company Inc., 1992

[15] B. Myers, "User-Interface Tools: Introduc -tion and Survey", IEEE Software, pp. 15-23, January 1989

[16] J. Raumbaugh et al., Object-Orented Modeling and Design Prentice-hall, Inc., 1991

[17] S. Rimmer, Graphical User Interface Programming, WINDCREST/McGRAW-HILL, 1991

[18] S. Shaler and S. Meller, Object-Oriented Systems Analysis: Modeling the World in Data, Englewood, Cliffs, NJ: Yourdon Press, 1988

저자소개

조대호

1983 성균관대학교 전자공학과 학사
1987 알라바마대 전자공학과 석사
1993 아리조나대 전자 및 컴퓨터공학 박사
1993~1995 경남대학교 전자계산학과 전임강사
1995~현재 성균관대학교 전기전자 컴퓨터공학부 부교수
관심분야 : 모델링 시뮬레이션, 공장 자동화, 지능 제어, ERP

김은하

1998 대전대학교 정보통신공학과 학사
1998년~현재 성균관대학교 전기전자 및 컴퓨터공학부 석사과정
관심분야 : 시뮬레이션 방법론, 자동화 시스템