한국시뮬레이션학회 논문지
Vol. 21, No. 4, pp. 35-46 (2012. 12)
http://dx.doi.org/10.9709/JKSS.2012.21.4.035

C-DEVS형식론을 이용한 실시간 이산사건 제어시스템의 논리 해석 기법

송해상[1†] · 김탁곤[2]

Logical Analysis of Real-time Discrete Event Control Systems Using Communicating DEVS Formalism

Hae Sang Song · Tag Gon Kim

ABSTRACT

As complexity of real-time systems is being increased ad hoc approaches to analysis of such systems would have limitations in completeness and coverability for states space search. Formal means using a model-based approach would solve such limitations. This paper proposes a model-based formal method for logical analysis, such as safety and liveness, of real-time systems at a discrete event system level. A discrete event model for real-time systems to be analyzed is specified by DEVS(Discrete Event Systems Specification) formalism, which specifies a discrete event system in hierarchical, modular manner. Analysis of such DEVS models is performed by Communicating DEVS (C-DEVS) formalism of a timed global state transition specification and an associated analysis algorithm. The C-DEVS formalism and an associated analysis algorithm guarantees that all possible states for a given system are visited in an analysis phase. A case study of a safety analysis for a rail road crossing system illustrates the effectiveness of the proposed method of the model-based approach.

Key words : Logical Analysis, Communicating DEVS formalism, Real-time Discrete Event Systems, Safety Analysis

요 약

실시간 시스템의 복잡도가 증가함에 따라 임시방편적 시스템 해석 방법은 시스템 동작 영역 전체를 완전하게 분석하는 데는 한계가 있다. 모델링을 기반으로 한 정형 기법은 그러한 한계점을 극복 할 수 있다. 본 논문은 모델 기반 정형 기법을 이용하여 실시간 시스템의 안전성 및 필연성 등과 같은 논리적 타당성을 이산 사건 모델 수준에서 분석하는 방법을 제안한다. 먼저, 분석 대상 실시간 시스템은 이산사건 수준에서 계층적으로 모듈화하여 모델을 명세하는 수학적 형식론인 DEVS (Discrete Event Systems Specification) 형식론으로 기술된다. 다음으로, 기술된 DEVS 모델은 시간 명세가 포함된 전역 상태 공간을 표현하는 C-DEVS (Communicating DEVS) 형식론으로 표현한 후 C-DEVS 형식론의 해석 알고리즘을 통해 시스템 동작을 분석된다. 제안된 C-DEVS 형식론 및 해석 알고리즘은 주어진 시스템의 동작 특성을 분석하는 과정에서 시스템의 상태 공간을 완전하게 빠짐없이 탐색하는 것을 보장한다. 간단한 건널목 제어 시스템의 안전성 분석 사례 연구를 통하여 제안된 모델 기반 해석 기법의 효율성을 예시 하였다.

주요어 : 논리해석, C-DEVS 형식론, 실시간 이산사건 시스템, 안전성 해석

1. 서 론

컴퓨터기반 제어는 현대의 교통제어, 비행기, 원자력발전소, 병원의 환자보호 시스템 등 실시간 제어 시스템들에서 폭넓게 활용되고 있다[1]. 그러한 시스템들에서는 제어의 실패가 사람의 생명과 재산에 막대한 파급효과를 미치기 때문에 안전성 및 필연성 등의 문제가 가장 중요한

접수일(2012년 8월 23일), 심사일(1차 : 2012년 10월 25일), 게재 확정일(2012년 11월 12일)
[1)] 서원대학교 컴퓨터공학과
[2)] KAIST 전기및전자공학과
주 저 자 : 송해상
교신저자 : 송해상
E-mail; hssong@seowon.ac.kr

문제 중의 하나로 대두되어 왔다. 그래서 컴퓨터 제어를 사용하는 그러한 실시간 시스템의 개발의 초기 단계에서 설계된 제어기의 안전성 등의 논리적 특성을 검증하는 것이 필요하며, 이러한 특성이 검증된 이후에 비로소 구현 단계에 들어가는 것이 일반적이다.

특히 실시간 제어시스템의 안전성은 제어 대상인 플랜트와 컴퓨터 제어가 연동되어 동작할 때 어떠한 경우에도 심각한 결과를 초래하는 단 하나의 나쁜 상태에도 들어가지 않는 것으로 정의된다. 따라서 안전성 해석은 그 시스템이 도달하게 될 모든 상태를 생성하고 점검해 보아야 한다. 따라서 한 가지 상태 순서만을 생성하는 시뮬레이션으로는 안전성 해석을 할 수는 없다.

따라서 이러한 해석은 보통 도달성 분석을 통해 이루어지는데 이는 초기 상태로부터 직간접적으로 도달 가능한 모든 상태를 생성해내기 때문이다[1-4]. 실시간 시스템에 대한 도달성 분석은 시간 자체가 독립적인 연속변수이므로 이론적으로는 상태폭발을 피해가기는 어렵다는 것은 어쩔 수 없다.

일반적으로 다수의 컴포넌트로 이루어진 실시간 시스템의 도달성 분석은 컴포넌트간 시간이 개입된 상호작용 및 상태천이 규칙을 따라야 한다. 실시간 시스템은 어떤 상태에 머무를 수 있는 시간에 한정되어 있으며, 또한 컴포넌트간 메시지 통신을 통해 상호 동작하기 때문에 이로 인한 상태천이 규칙 또한 존재해야 한다. 특히 통신하는 과정에 있어서 메시지 손실이 있을 경우도 고려해야 한다. 따라서 이러한 도달성 분석을 위해서는 먼저 잘 정의된 의미를 가진 형식 모델을 사용하여야 하며, 또한 시간을 고려한 상태 천이와, 메시지 손실을 고려한 컴포넌트간 통신 규칙을 명세하고 있어야 한다.

안전성 분석을 위해서는 서로 다른 모델링 형식론을 기반하여 몇 가지 방법이 제시되어 왔다. 그 중 페트리네트(PN: Petri-Net)는 최근 몇 십년간 많은 응용분야에서 실시간 이산사건 시스템의 모델링과 안전성 분석에 사용되어 왔다[1,4]. 플래이스와 트랜지션, 그리고 토큰으로 구성된 PN은 필요에 따라 다양한 변종이 있어왔고 도달성 분석도 잘 정의되어 안전성 분석에 활용되어 왔으나, 결정적으로 안전성해석이 완료된 PN 모델을 컴퓨터 소프트웨어로 구현하기에는 적절치 않았다. 이는 PN 모델이 상태기반이 아니기 때문에 PN모델을 상태변수 및 변수에 대한 연산에 기반한 컴퓨터 소프트웨어로 1:1 대응시켜 구현하기에는 적절치 않기 때문이다. RT-DEVS (Real-Time DEVS) 형식론은 DEVS를 확장하여 실시간 시스템 모델링 및 중단 없는 구현에 적용하기 위한 것이다[2]. 이 모델링 형식론은 시간을 고려한 도달성 분석 기법인 TBA (Timed Behavior Analysis)를 제공하므로 안전성분석에 적용할 수 있는 대안이다. 하지만, 시간을 구간으로 지정할 수 있는 장점에도 불구하고 메시지의 손실을 고려하고 있지 않다. 시간 오토마타 (TA: Timed Automata)은 오토마타에 시간 정보를 부가시킬 수 있도록 확장한 형식론으로 시간영역 (region) 분석을 통해 도달성분석을 할 수 있으나 하나의 컴포넌트의 동작을 모델링하는 오토마타의 태생적 불편함을 가지고 있다[3].

본 논문은 이산사건시스템을 시스템 이론에 입각하여 계층적이며 모듈 형태로 기술할 수 있는 잘 정의된 의미론을 가지고 있는 DEVS (Discrete Event Systems Specification) [5] 모델에 대한 안전성 분석 기법을 제안하고자 한다. DEVS는 Fig. 1의 우측과 같이 그동안 모델링 및 시뮬레이션은 많이 활용되어 왔으나, 좌측의 논리적인 해석에는 좀처럼 적용이 활성화되지 않았다[9,10]. 그동안 시간을 고려하지 않은 무시간 해석 기법[6]과 시간을 고려한 RT-DEVS해석기법[2] 등이 있었으나 시간을 해석하지 못하거나 메시지손실을 처리하지 못하는 논리해석 기법이었다. 따라서 본 논문은 기존의 연구[7,8]를 확장하여 DEVS 모델에 대한 상호 작용 규칙 및 분석 기법을 DEVS 형식론의 의미론에 합당하게 정의하고, 주어진 DEVS 모델이 나타낼 수 있는 모든 가능한 행위를 분석할 수 있도록, 컴포넌트의 상호작용 규칙을 명시한 C-DEVS (Communicating DEVS)

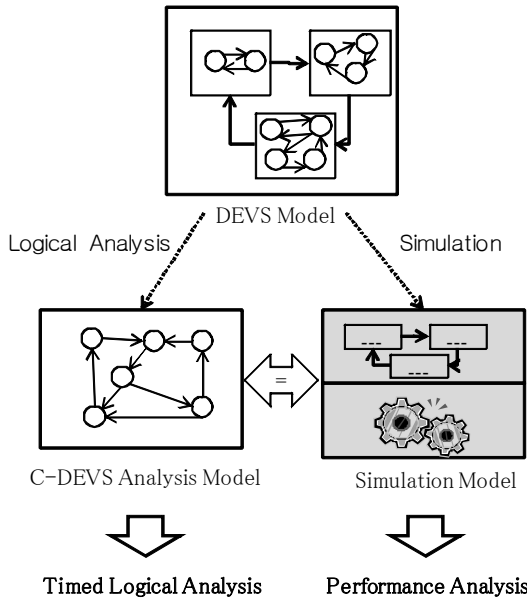

Fig. 1. Unified DEVS model analysis and simulation methodology.

형식론을 명확히 정의하고, 이 규칙을 적용해서 다수의 원자모델의 복합 행위 모델을 만들어낼 수 있는 알고리즘을 제시하고자 한다. 또한, C-DEVS 형식론이 동일한 DEVS 모델의 시뮬레이션과 일치하는 규칙을 가지는지 검증하기 위해서 모델에 대한 시뮬레이터를 구현하고, 생성된 상태와 사건, 그리고 시간을 해석적인 방법으로 얻어진 결과와 일치함을 보여서, C-DEVS 형식론의 규칙이 DEVS의 원래의 규칙을 충실히 만족함을 보이고자 한다. 또한 제안된 방법론을 사용하여 간단한 실시간 제어 시스템 모델에 적용하여 안전성과 필연성을 검증해 보이고자 한다. 본 논문은 또 다른 안전성 등의 논리 분석 방법이기는 하지만, M&S에서 많이 사용되는 DEVS 형식론의 동적인 의미론을 완전히 수용한 것으로는 최초의 것이다.

이 논문은 다음과 같이 구성되어 있다. 다음 절에서는 DEVS 형식론과 C-DEVS 형식론을 정의하고, C-DEVS 모델을 생성할 수 있는 알고리즘을 제안한다. 3절에서는 제안된 방법론을 적용하여 간단한 건널목 제어 모델에 대해서 안전성을 분석한다. 4절에서는 모델의 시뮬레이션 결과와 C-DEVS 모델을 검증한 결과를 비교 제시한다. 5절에서는 요약과 향후 연구방향으로서 결론을 맺고자 한다.

2. C-DEVS 형식론 및 해석기법

2.1 DEVS Formalism

문헌에서 잘 알려진 바와 같이 DEVS 형식론은 이산사건 시스템을 단위 컴포넌트의 행위를 기술하는 원자모델 형식론과 모델들이 계층적으로 사건에 의해 결합되는 구조를 나타내는 결합모델에 대한 형식론으로 구성되어 있다[5]. 원자모델은 다음과 같이 3개의 집합과 4개의 함수로 구성한다.

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

여기서 집합 X는 입력사건집합, Y는 출력사건집합, S는 상태집합이다. 원자모델의 상태와 경과된 시간의 쌍의 집합은 $Q=\{(s,e)|s\in S, 0\le e\le ta(s)\}$, 즉 상태와 그 상태에서의 경과시간의 쌍으로 정의하며 이것을 시간상태 또는 전체상태라고 한다. 4개의 함수는 다음과 같은 제약사항을 갖는다.

$\delta_{ext}: Q\times X\to Q$: 외부상태천이함수, 여기서 $\delta_{ext}(s,e,x)=(s',e'),\ e<ta(s), e'=0.$

$\delta_{int}: Q\to Q$: 내부상태천이함수, 여기서 $\delta_{int}(s,e)=(s',e'),\ e=ta(s), e'=0.$

$\lambda: Q\to Y$: 출력함수, 여기서 $(s,e)\in Q,\ e=ta(s).$

$ta: S\to R_0^+$: 시간전진함수, 여기서 R_0^+ 0이상의 양의 실수의 집합

원자모델에서는 두 종류의 상태천이를 한다. 내부상태천이는 현재 상태의 시간전진, 즉, 머무를 수 있는 시간까지 입력사건이 발생되지 않았을 때 스스로 천이하는 것으로서 출력사건이 λ함수로 정의되어 있으면 출력사건을 내보내고, δ_{int}함수에서 정의된 상태천이를 한다. 외부상태천이는 현재 상태 s에서 시간전진 $ta(s)$이내에 입력사건 x가 도달하면 $\delta_{ext}(s,e,x)=(s',0)$에서 정의된 대로 새로운 상태로 바꾼다. 위의 제약사항들은 시간상태에 대해서 정의하였음을 알 수 있다. 하지만 실제 시스템명세를 할 때에는 시간적인 의미론을 위의 정의처럼 함축된 것으로 가정한 후 3 개의 함수는 $\delta_{ext}: S\times X\to S$로, $\delta_{int}: S\to S$로, $\lambda: S\to Y$로 시간을 명시하지 않고 명세하여도 위의 정의와 동일한 것으로 간주한다. 이 때 시간은 시간전진함수 $ta: S\to R_0^+$에서만 정의한다.

시간을 해석할 경우에는 시간상태 $q=(s,e)\in Q$ 를 가지고 표기하는데, 이는 어떤 상태 s에서 머무른 시간 , 즉 경과시간 e의 쌍으로 표시한다. 해석을 위해서 편의상 경과시간보다는 잔여시간을 사용하기도 하는데 이는 $\sigma=ta(s)-e$를 나타낸다. 상태와 경과시간 대신 상태와 잔여시간의 쌍을 잔여상태라고 부르며 시간상태와 잔여상태를 동등하게 다루기로 한다.

DEVS 결합모델의 형식론은 하나의 결합모델 내에 존재하는 모델들의 출력사건들과 입력사건들을 연결해 주는 것으로, 본 논문의 초점이 아니므로 간략화 시켜 사건의 이름이 같으면 모델들이 서로 결합되어 있다고 가정한다. 또한 계층적인 결합모델들은 평탄하게 만들어서 하나의 결합모델은 단지 원자모델들의 집합으로 생각하기로 한다.

2.2 C-DEVS 형식론

C-DEVS(Communicating DEVS) 형식론은 DEVS 원자모델들의 집합으로 이루어진 하나의 이산사건 시스템의 행위를 시뮬레이션 과정 없이 초기 상태로부터 도달할 수 있는 모든 가능한 상태천이를 분석해 볼 수 있도록 정의한 것이다. 구체적으로 C-DEVS 는 임의의 N개의 원자 DEVS 모델들 간의 상호작용 기제를 수학적으로 정의한

것이며, 비동기, 동기, 그리고 사건손실의 세 가지의 상호작용을 정의한다. 이것은 RT-DEVS [2]에서 정의한 약동기 (Weak Synchronization)에서 정의한 앞의 두 경우에 추가하여 사건이 전달되지 않고 손실되었을 경우까지 포함하는 것으로 확장한 것이다. 또한, RT-DEVS에서는 시간전진을 구간으로 나타내었지만, C-DEVS에서는 시점으로 나타낸다.

임의의 N개의 모델로 확장하기에 앞서 두 개의 모델로 이루어진 다음과 같은 시스템 $M=\{\mathrm{AM_i},\mathrm{AM_j}\}$, $AM_k=<X_k,Y_k,S_k,\delta_{\mathrm{ext,k}},\delta_{\mathrm{int,k}},\lambda_k,ta_k>, k=i,j$에 대해 상호작용을 정의하고자 한다. 두 원자모델이 현재 시각에 $q_i=(s_i,e_i)$ 및 $q_j=(s_j,e_j)$의 상태라고 가정한다. 시스템 M의 C-DEVS 모델은 다음과 같이 정의된다.

$$CDEVS(M)=AM_i\|AM_j=<E,Q,T,ta,M>$$

E: 이벤트 집합
Q: 합성시간상태 집합
T: 합성 원자 모델의 상태 천이 관계
ta: 합성 상태의 시간 전진 함수
$M=\{\mathrm{AM_i},\mathrm{AM_j}\}$: 원자모델 집합

단, 위 항목은 다음과 같은 제약조건을 만족한다.

$E=(X_i\cup Y_i\cup\{\epsilon\})\times(X_j\cup Y_j\cup\{\epsilon\})$, ϵ은 무사건, $Q\subseteq S\times R^2$, 여기서 $S=S_i\times S_j$는 무시간 합성상태이며, $R^2=R\times R$은 각 모델의 경과시간의 벡터.

$T\subseteq Q\times E\times Q$,

$ta: Q\rightarrow R$

여기서, Q는 합성시간상태로서 각 원자모델의 시간상태의 곱으로 정의된다. R은 양의 실수의 집합이다. Q는 $Q\subseteq Q_i\times Q_j=(S_i\times R)\times(S_j\times R)=(S_i\times S_j)\times(R\times R)=S\times R^n$로 나타낼 수 있다. 여기서 R^n은 경과시간의 벡터로서 n은 원자모델의 개수이다. $S=S_i\times S_j$를 우리는 무시간 합성상태 또는 간단히 합성상태라고 한다. 무사건 (Null Event) ϵ은 사건이 일어나지 않은 것을 의미한다.

C-DEVS 형식론은 위 두 모델의 사건을 통한 상호작용을 합성 원자모델의 상태천이관계 T 및 시간전진 함수 ta로서 정의하며 상태 천이는 각 모델의 현재 시간상태 $q_i=(s_i,e_i)$ 및 $q_j=(s_j,e_j)$에 대해서 다음과 같은 3가지 규칙을 따른다. 여기에서 수학적인 표기의 편리함을 위해 함수가 $\delta_{int,i}(s_i,ta(s_i))=s'_i$로 정의되었을 때 $(s_i,s'_i)\in\delta_{int,i}$로 표시하도록 하자(다른 함수도 마찬가지로 적용한다). 최초 합성시간상태의 시간전진 값은 잔여시간의 최소값, 즉 $ta(q_i,q_j)=\min\{\sigma_i,\sigma_j\}$, 여기서 $\sigma_i=ta_i(s_i)-e_i$, $\sigma_j=ta_j(s_j)-e_j$ 로 정의한다.

Rule 1 (비동기 천이): 잔여시간이 $\sigma_i\le\sigma_j$일 때, 만약 $(s_i,s'_i)\in\delta_{int,i}$, $(s_i,!a)\in\lambda_i$이며 $(s_j,?a,s'_j)\notin\delta_{ext,j}$ 일 경우에 AM_i 단독으로 비동기 천이.

(a) 상태천이관계

$$((q_i,q_j),(!a,\epsilon),(q_i',q_j'))\in T,$$
$$q_i'=(s_i',0),\ q_j'=(s_j,e_j+\sigma_i).$$

(b) 시간전진

$$ta(q_i',q_j')=\min\{\sigma_i',\sigma_j'\}=\min\{ta_i(s_i'),\sigma_j-\sigma_i\}$$

Rule 2 (동기 천이) : 잔여시간이 $\sigma_i\le\sigma_j$일 때, $(s_i,s'_i)\in\delta_{int,i}$이고 $(s_i,!a)\in\lambda_i$이며, $(s_j,?a,s'_j)\in\delta_{ext,j}$ 일 경우 사건의 동기화를 통한 두 모델 AM_j과 AM_j이 동시 천이.

(a) 상태천이관계

$$((q_i,q_j),(!a,?a),(q_i',q_j'))\in T,$$
$$q_i'=(s_i',0),\ q_j'=(s_j',0).$$

(b) 시간전진

$$ta(q_i',q_j')=\min\{\sigma_i',\sigma_j'\}=\min\{ta_i(s_i'),ta_j(s_j')\}$$

Rule 3 (손실 천이) : 잔여시간이 $\sigma_i\le\sigma_j$일 때, 만약 $(s_i,s'_i)\in\delta_{int,i}$, $(s_i,!a)\in\lambda_i$이며 $(s_j,?a,s'_j)\in\delta_{ext,j}$ 로 사건 동기화 조건을 만족하나 사건이 손실되었을 경우, AM_i 단독으로 천이.

(a) 상태천이관계

$$((q_i,q_j),(!a,\epsilon),(q_i',q_j'))\in T,$$
$$q_i'=(s_i',0),\ q_j'=(s_j,e_j+\sigma_i).$$

(b) 시간전진

$$ta(q_i',q_j')=\min\{\sigma_i',\sigma_j'\}=\min\{ta_i(s_i'),\sigma_j-\sigma_i\}.$$

위 상호작용 규칙들을 살펴보면 다음과 같다. 먼저 Rule 1은 비동기 형태의 상호작용으로 가장 작은 잔여시간을 갖는 원자모델이 출력사건을 내고, 이 출력사건을 다른 모델들이 받을 상태에 있지 아니할 때의 경우로서 출력사건을 내는 원자모델만 상태를 천이하고 다른 모델들은 경과시간만 증가하는 것을 의미한다. Rule 2는 동기식 천이로 가장 작은 잔여시간을 갖는 원자모델이 출력사

건을 내고, 이를 받을 수 있는 다른 모든 원자모델들이 동시에 천이하는 것이다. 이 동기에 참여하는 모든 모델들의 경과시간은 0으로 초기화된다. 나머지 모델은 Rule 1을 따라 경과시간만 증가한다. 마지막으로 Rule 3는 손실천이로서 가장 작은 잔여시간을 갖는 원자모델이 출력사건을 냈지만 손실되어 이를 받을 수 있는 원자모델들조차 천이에 참여할 수 없는 경우이다. 이 경우에는 Rule 1처럼 출력사건을 내는 모델만 상태를 바꾸고 나머지는 현재 상태에서 경과시간만 증가한다.

이러한 세 가지 규칙은 단지 두 개의 모델에 대해서 정의하였지만, 출력을 내는 하나의 모델과 다수의 나머지 모델로 쉽게 확장할 수 있음을 알 수 있다.

2.3 C-DEVS 모델 구성 알고리즘

C-DEVS 형식론에서는 다만 모델들간의 상호작용만 정의한 것이기 때문에 주어진 원자모델의 집합을 하나의 C-DEVS 모델로 만드는 알고리즘이 필요하다. 주어진 초기 복합시간상태로부터 시작하여, C-DEVS 상호작용의 3가지 규칙을 적용해 가면서 도달하는 복합시간상태를 얻어가는 방식으로 알고리즘은 구성되어 있다. 복합시간상태는 경과시간보다는 동등한 표현력을 가지는 잔여시간으로 표현한 복합잔여상태를 사용한다. 복합잔여상태에서 가장 적은 잔여시간을 갖는 원자모델을 선택하여 나머지 모든 모델에 대해서 세 가지 규칙을 적용한다. 이를 통해 도달하는 복합잔여상태로부터 다시 반복적으로 이 규칙을 적용한다. 한번 점검한 상태는 더 이상 점검하지 않는다. 알고리즘의 입력은 원자모델의 집합 M과 초기 복합시간상태이며, 출력은 C-DEVS의 상태천이 모델이다. 이를 알고리즘으로 정의하면 다음과 같다.

Algorithm 1. C-DEVS_CONSTRUCTION(M, q_0)

Input: n개의 원자모델들의 집합 M 및 초기 복합시간 상태 q_0.

Output: M에 대한 C-DEVS 모델, C-DEVS(M).

Begin Procedure

1. $q_0 \in Q$는 초기 복합시간상태, $Q_{done} \subseteq Q$는 완료상태 집합, 그리고 $Q_{exam} \subseteq Q$ 는 방문할 상태집합이라고 하고 다음과 같이 초기화한다.
$$Q_{done} = \emptyset,\ T = \emptyset,\ Q_{exam} = \{q_0\}.$$
2. 임의의 $q = (q_1, q_2, \cdots, q_n) \in Q_{exam}$를 선택한다.
3. 선택한 복합시간상태 q의 시간전진 값을 구성 원자모델들의 잔여시간 중 가장 작은 잔여시간으로 지정한다. 즉, $\sigma_k = ta_k(s_k) - e_k$ 라 할 때, $ta(q) = \sigma_{\min} = \min_{k=1\cdots n}\{\sigma_k\}$.
4. q에 대해서 $\sigma_i = ta_i(s_i) - e_i = \sigma_{\min}$인 임박한 모델 i의 시간상태 q_i 및 다른 모든 모델의 시간상태 $q_j, j \neq i$ 에 대해서 천이 규칙을 적용하여 만들어진 상태천이를 T에 추가한다. 그 상태천이의 목적상태 $q' = (q_1', q_2', \cdots, q_n')$, $(q, e, q') \in T$를 방문할 상태집합에 추가한다. 단, $q' \notin Q_{done}$이면 추가하지 않는다. 즉,
$$Q_{exam} = Q_{exam} \cup \{q'\} \text{ if } q' \notin Q_{done}.$$
5. 다른 임박한 모델이 있다면 단계 4를 반복한다.
6. 검사된 상태 q를 완료상태집합으로 옮긴다. 즉, $Q_{exam} = Q_{exam} - \{q\}$, $Q_{done} = Q_{done} \cup \{q\}$.
7. $Q_{exam} = \emptyset$이 될 때까지 단계 2-6을 반복한다. (필요에 따라 선지정한 조건이 되면 종료할 수 있다.)
8. 7단계까지 완료되면 주어진 M에 대해서 $CDEVS(M) = < E, Q, T; ta, M >$이 얻어졌다.

End Procedure ■

제안된 절차는 복합시간상태 Q가 복합상태와 경과시간의 벡터의 쌍이어서 이론적으로 시간이 무한이기 때문에 천이관계가 한정적이라는 보장을 하지 않는다. 따라서 적절한 조건을 만족하면 중단하거나, 논리적 특성을 만족하면 중단하도록 하여야 한다. 이러한 상태폭발의 문제는 본 논문의 주제가 아니므로 다루지 않으나 저자들의 경험상 실제 문제에 있어서는 주기적으로 시스템이 동작하도록 설계되기 때문에 한정적인 경우가 존재한다고 할 수 있다.

3. 적용사례: 건널목 제어시스템(RRC)의 안전성 및 필연성 해석

3.1 문제의 정의

열차와 차가 교차하는 간단한 기차건널목 제어 시스템(RRC: Railroad Crossing Control System)을 생각해보자. 치명적인 사고를 방지하기 위해서는 기차의 접근을 감지하고 차단기와 신호를 통해 차량이 건널목을 통과하지 못하게 해야 하며, 기차가 통과한 후에는 차단기를 올려 차량이 궁극적으로 통과할 수 있게 제어하여야 한다. 이를 위한 컴퓨터 기반 제어기는 다음과 같은 두 개의 목표를 가져야 한다.

(1) **(안전성)** 기차의 위치를 감지해서 사고가 나지 않도

록 올바른 시간에 올바른 순서로 차단기를 조작해야 한다.

(2) **(필연성)** 건널목을 지나려고 하는 차량은 한정된 시간 내에 언젠가는 건널목을 통과하여야 한다.

다른 건널목제어 문제와는 달리 본 논문에서는 제어시스템의 컴포넌트 간의 송수신 제어 메시지가 열악한 통신환경, 소프트웨어 버그, 또는 어떠한 인적 오류에 의해서 손실될 수 있다고 가정한다. 목표를 달성하기 위한 강건한 제어 시스템을 얻기 위해서는 먼저 건널목 플랜트 시스템을 모델링하고, 목표를 만족하는 제어기를 설계하여야 한다. 설계된 제어기를 플랜트와 연동하였을 때 위의 안전성과 필연성을 논리적으로 만족하는지를 검증하여야 한다. DEVS는 이러한 수준의 실시간 시스템을 표현하기에 충분하며, 표현된 모델을 시뮬레이션 엔진을 붙여 시뮬레이션할 수 있을 뿐만 아니라 본 논문과 같이 해석하기에 적절한 형식론임을 보이고자 한다.

3.2 RRC DEVS Models

건널목 제어 시스템은 Fig. 2처럼 TRAIN, GATE 및 CONTROLLER의 세 개의 원자모델이 결합되어 동작하는 모델로 표현할 수 있다. 각 원자모델에서 타원형은 상태를 나타내며, 타원 상부에는 상태의 이름, 하부에는 그 상태에서 머무를 수 있는 시간을 표시한다. 초기 상태는 화살표로 적시하였다. 예를 들어 Fig. 2 (a)에서 초기 상태는 Trav 이고 그 상태에서 TRAIN은 300초간 머무른다. 내부 상태천이는 점선 화살표로 출발상태와 도착상태를 표기하였고, 화살표 위에는 이 때 출력할 사건을 사건 이름 앞에 '!' 접두어를 붙여 나타내었다. 외부 상태천이는 실선 화살표로 표기하였고 천이를 일으키는 입력사건은 '?'접두어를 붙여 명확히 하였다. 앞에서 언급한 것처럼 사건의 이름이 같으면 두 모델이 그 사건에 의해서 연결되었으며 계층적인 결합모델은 계층을 없애서 평탄한 모델로 변환하였다고 가정한다. 그림으로 표현된 위 DEVS 모델은 다음과 같이 DEVS 형식론에 의해서 수식적으로 나타낼 수 있다.

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

X={}, Y={!ap, !ex, !in}, S={Trav, Appr, Passing},
δ_{int}(Trav) = Appr, δ_{int}(Appr) = Passing,
δ_{int}(Passing) = Trav,
λ(Trav) = !ap, λ(Appr)=!in, λ(Passing) = !ex,
ta(Trav) = 300, ta(Appr)=30, ta(Passing) = 30.

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

X={?dn,?up}, Y={},
S={Open, Lower, Closed, Raise},
δ_{int}(Lower) = Closed, δ_{int}(Raise) = Open,
δ_{ext}(Open,?dn) = Lower, δ_{ext}(Closed,?up)= Raise,
ta(Lower) = 5, ta(Raise)=5.

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

X={?ap,?in,?ex}, Y={!dn,!up},
S={TU,AU, AD, PD, TD},
δ_{int}(AU) = !dn, δ_{int}(TD) = !up,
δ_{ext}(TU,?ap) = AU, δ_{ext}(AD,?in)= PD,
δ_{ext}(PD,?ex)= TD,
ta(AU) = 3, ta(TD)=2.

모델에서 본 바와 같이 기차의 모델은 제어기 입장에서는 센서를 통해 추상화되고, 차단기에 명령을 내려 안전성과 필연성을 보장해야 하는 형태로 구성되어 있다. 열차는 반복적으로 건널목에 도달한다고 가정하였고, 건널목에서 일정 거리 전에 있는 감지기에 의해서 접근 사건 (ap)을 발생시키고, 통과하기 시작하면서 in 사건과, 후미가 빠져나가면서 ex 사건을 발생시키는 것으로 추상화하였다. 제어기는 차단기를 올리라는 제어 명령 up과 내리라는 dn 명령을 제어기로부터 받는다. 제어기 모델은 초기에 TU 상태에서 ap 사건을 Train 모델로부터 받으면 AU상태로 천이하고 3초 이내에 게이트에 dn 명령을 내

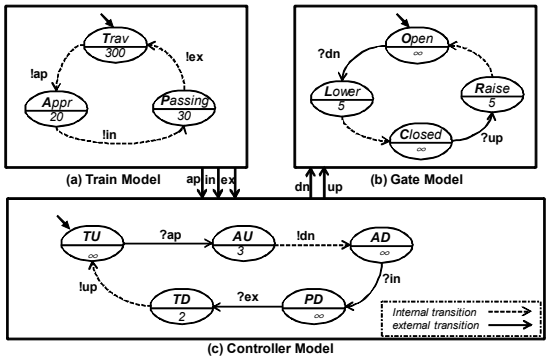

Fig. 2. Rail-Road Crossing control model: TRAIN, GATE, and CONTROLLER model.

리고 AD상태로 간다. 여기에서 in 사건을 받아 기차가 건널목 내에 있음을 감지하고, ex 사건을 받아 기차가 건널목을 완전히 통과했음을 인지한 후 up 명령을 내린다. 이러한 구성 모델간의 사건은 손실될 수 있다고 가정하고 안전성과 필연성을 해석하고자 한다.

3.3 RRC C-DEVS: 사건 무손실 해석

안전성 해석은 C-DEVS 모델을 만들어내는 과정에서 하거나, 완전히 C-DEVS 모델을 얻은 후에 실시할 수 있다. 안전성 해석을 위해서는 먼저 안전하지 않는 상태 집합인 위험상태집합을 정의하여야 한다. 즉, 시스템이 도달하면 치명적인 위험이 발생하는 상태들의 집합을 지칭하며, 위의 예시에서는 차량과 기차가 동시에 건널목에 있는 경우, 즉 기차가 Passing 상태이고 건널목이 Open 상태인 경우가 될 것이며 이를 기차와 차단기의 복합상태로 표현하면 (Passing, Open)이 된다. 따라서 위험상태집합은 다음과 같은 집합 B로 표현되며, 시스템이 이 집합에 속한 하나 이상의 상태에 도달하면 안전성을 만족하지 않는다.

• B={(Passing, Open)}.

필연성 (Liveness)은 상태가 아니라 순서로 정의된다. 즉, 차단기가 닫혔으면 언젠가는 반드시 열려야 한다는 것은 기차와 차단기의 복합상태의 순서로 다음과 같이 Temporal Logic과 유사한 수식으로 표현할 수 있다.

• 차량: (-, Closed) ⇒ ◇ (-, Open) ∧
• 기차: (Appr,-) ⇒ ◇ (Passing, -)

여기서 ⇒ 는 "의미한다 (imply)"는 기호이고 ◇는 언젠가는 만족해야한다는 기호이다. 즉, 차량의 입장에서는 차단기가 닫혔으면 반드시 언젠가는 열려야 한다는 뜻이고, 기차의 입장에서는 접근했으면 반드시 언젠가는 통과해야 한다는 뜻이다.

Fig. 3은 사건손실이 없다는 가정 하에 제안된 알고리즘 1을 적용하여 무손실 C-DEVS 모델을 도식화한 것으로 앞의 그림과 같이 타원은 상태이고, 실선은 상태천이이다. C-DEVS의 세 가지 상호작용 규칙 중 Rule 1과 Rule 2만을 적용하였으며, Rule 3는 적용하지 않은 것이다. 초기 복합시간상태는 ((T,O,TU), [0,0,0])) 또는 동등한 잔여시간으로 표기할 경우 ((T,O,TU), [300,∞,∞])인데 이는 기차가 여행 중인 Trav, 차단기는 열린 상태 Open, 그리고 제어기는 기차의 접근을 기다리는 TU 상태인 (T,O,TU) 복합상태이고, 각 모델의 경과시간은 0임을 의미하며, 잔여시간은 시간전진함수에 의해 시간전진에서 경과시간을 뺀 값, 즉 [300,∞,∞]임을 의미한다. 즉, Train은 300초를 기다리고, 나머지 모델은 무한대의 대기상태임을 나타낸다.

위 복합시간상태의 시간전진은 각 모델의 잔여시간의 최소값인 300이며, 300초 이후에 Train이 !ap 출력사건을 내고, 제어기가 ?ap 사건을 받아서 동기식 상태천이를 하여 (A,O,AU)[20,∞,3]으로 천이하고 도달한 상태의 시간전진은 최소인 3초가 된다. 이러한 방식으로 초기상태에서 도달할 수 있는 모든 상태와 상태천이를 구할 수 있다. C-DEVS 모델은 다음과 같이 수식으로 표현할 수 있다.

$$
\begin{aligned}
&CDEVS(M) = < E, Q, T, ta, M >\\
&\quad M = \{ Train, Gate, Controller \}\\
&\quad E = \{(!ap,\epsilon,?ap),(\epsilon,?dn,!dn),(\epsilon,\epsilon,\epsilon),(!in,\epsilon,?in),\\
&\qquad\qquad (!ex,\epsilon,?ex),(\epsilon,?up,!up),\ldots\}\\
&Q = \{ < (T,O,TU),[300,\infty,\infty] >,\\
&\qquad < (T,O,TU),[293,\infty,\infty] >,\\
&\qquad < (A,O,AU),[20,\infty,3] >,\\
&\qquad < (A,L,AD),[17,5,\infty] >,\cdots\}
\end{aligned}
$$

T는 위 그림에서의 상태천이 및 사건 순서쌍으로 이루어진 원소의 집합으로 나타내며 예를 들어 (<(T,O,TU), [300,∞,∞]>, (!ap, ε,?ap), <(A,O,AU),[20,∞,3]>)은 T의 원소이다. 시간전진함수는 집합 Q의 모든 원소에 대해서 잔여시간의 최소의 값으로 정의된다. 예를 들어, ta(<(T,O,TU), [300,∞,∞]>) = 300이다. 주목할 것은 복합상태가 같더라도 각 모델의 경과시간이 달라지면 다른 복합시간상태로 취급되는 것으로, 이는 시간이 있는 해석

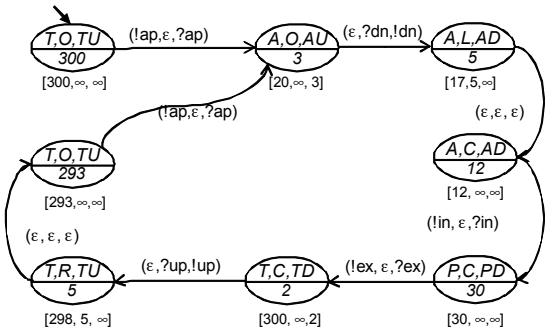

Fig. 3. C-DEVS analysis without message loss.

이기 때문에 당연한 것이다.

안전성해석은 이러한 과정에서 위험상태집합에 속한 상태에 도달하는 지를 판단하면 된다. 보통 이러한 연산을 모델검사(Model Checking)라고 한다. Fig. 3에서는 초기상태로부터 도달상태를 만들어가는 과정에서 모델검사가 이루어지며 집합 B에 속한 상태에 절대 도달하지 않았으므로 본 제어기는 안전성을 만족한다고 결론을 낼 수 있다.

필연성 해석은 상태의 순서에 대해서 이루어지며, C-DEVS 모델의 도달성 분석을 하면서 도달하는 상태를 인식하는 상태순서 인식기 (Recognizer)로 만들 수 있다. 위의 차량과 기차에 대한 상태 순서에 의해 2개의 상태 인식기를 동작시킨다. 예를 들어 차량에 대한 상태인식기는 먼저 (-, Closed) 상태에 도달하면 이어지는 상태들이 (-, Open)에 도달하는지를 검사하다가 도달하게 되면 인식기를 종료시킨다. 모든 상태순서 인식기들이 종료되었다면 해당되는 상태의 순서가 생성되었음을 의미한다. Fig. 3에서는 차량의 경우에 (A,C,AD)상태에서 차단기가 Closed 상태에 도달하였으므로 이어서 도달하는 상태들을 모두 받아들인 결과 (T,O,TU)상태에서 Open이 되었으므로 차량의 경우에 인식이 올바로 완료되었다. 또한, 기차의 경우에는 (A,O,AU)상태에서 처음 (Appr,-)를 만났으며, (P,C,PD)에서 기차가 Passing상태에 도달하였다. 따라서 두 개 모두 인식이 완료되었으므로 두 조건이 AND 조건을 만족하므로 필연성을 만족한다고 결론을 낼 수 있다.

시간적인 측면에서 볼 때, 차단기가 내려간 후 올라갔을 때까지의 시간흐름은 (A,C,AD) ···→ (T,O,TU)상태에 도달하기 까지 상태의 순서를 거칠 때 소요되는 시간을 계산하면 된다. 즉, 다음과 같이 84초가 소요된다.

- (A,C,AD) 17초 → (P,C,PD) 30초 → (T,C,TD) 2초 → (T,R,TU) 5초 → (T,O,TU) 도달: 84초.

RRC 모델의 행위는 전체로 사이클을 이루면서 동작하는데, 초기의 과도기를 제외하고 안정적으로 동작할 때의 한 사이클이 걸리는 시간은 (A,U,AO)부터 (T,O,TU)로 이루어지는 사이클의 경우 355초가 소요된다.

결론적으로 사건손실이 없을 경우 위 제어기는 안전한 동작을 보장하며, 또한 필연적으로 차량이 제어기를 통과할 수 있으며 84초 정도 대기 시간이 있음을 알 수 있었으며 한 사이클 가운데 약 77%가 차량이 건널목을 사용할 수 있음을 알 수 있었다.

3.4 RRC C-DEVS: 사건 손실 해석

본 해석의 목적은 제어 메시지 혹은 사건 손실이 있을 경우 제어기를 포함한 전체 시스템이 어떻게 동작하는지를 분석하여 논리적 해석을 하는 것이다. Fig. 4는 Fig.

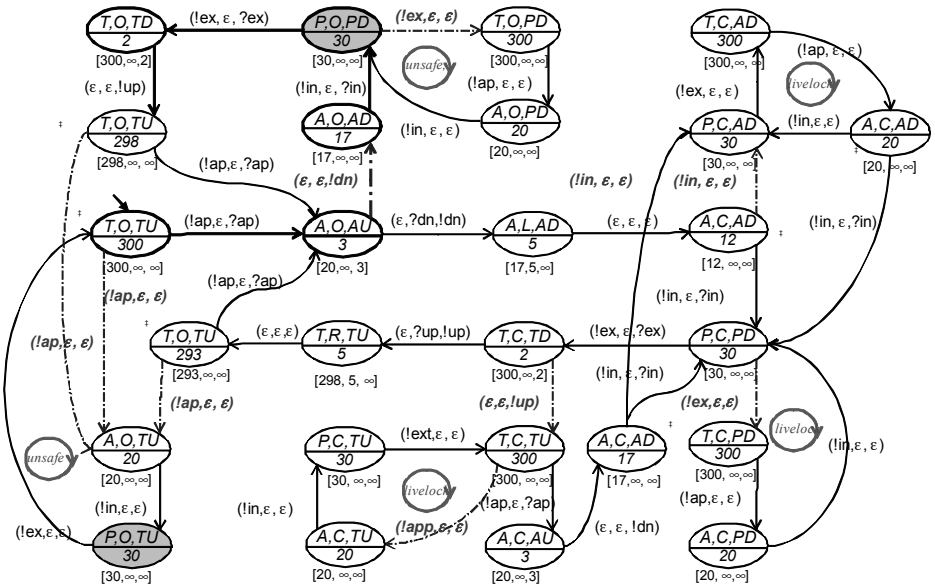

Fig. 4. C-DEVS analysis of RRC DEVS model with message loss, where the lost event set = {ap, in, ex, dn, up}.

2에서 보여준 결과와는 달리 구성요소 간에 송수신하는 모든 사건이 손실될 수 있다는 다른 한 극한 가정 하에 제안된 알고리즘으로 얻은 C-DEVS 모델을 그린 결과이다. 그림에서 사건손실이 일어나지 않을 때의 결과에 비해서 훨씬 더 많은 상태가 존재함을 알 수 있다.

RRC 모델은 복합시간상태 ((T,O,TU),[0,0,0])에서 시작한다. 즉 Train은 (T,0)이고 차단기는 (O,0) 그리고 제어기는 (TU,0)의 상태이다. 그림에서는 해석의 편의상 각 모델의 시간전진함수를 적용하여 위 복합시간상태와 동등한 복합잔여시간상태 ((T,O,TU), [300,∞,∞])로 표현하였다. 초기 복합상태의 시간전진은 구성원자모델의 잔여시간 중 가장 적은 잔여시간인 300이다. 그 시간이 지난 후 Train은 !ap 출력사건을 내고, Controller가 그것을 받은 복합 사건 (!ap,e,?ap)이 발생하고 Rule 2의 동기식 천이를 통해 ((A,O,AU),[20,∞,3])에 도달한다. 하지만 !ap 사건은 손실될 수 있으므로 Controller는 그것을 받지 못하는 (!ap,ε,ε) 사건이 발생될 수 있고 Rule 3에 의해서 ((A,O,TU),[20,∞,∞])로 천이할 수 있다. 후자의 경우에는 ((P,O,TU),[30,∞,∞])로 (!in, ε, ε) 사건에 의해서 천이할 수 있다. 복합상태 (P,O,TU)는 위험상태집합에 속하므로 안전성을 보장할 수 없다는 것을 알 수 있다. 이 상태에서 다시 (!ex,ε,ε) 사건이 발생하고 초기 상태로 돌아갈 수 있다. 사이클을 형성하였으므로 지속적으로 이러한 사건이 발생될 수 있음을 또한 알 수 있다. 따라서 !ap 사건은 절대로 손실되지 않아야 할 중요한 사건임을 찾아낼 수 있다. 좌측 상단에서도 (A,O,AU)상태에서 !dn 사건 즉, 차단기를 내리라는 명령이 손실됨으로서 궁극적으로 위험 상태에 도달함을 알 수 있다.

필연성의 측면에서 우측 상단을 살펴보면 다음과 같은 사이클이 존재함을 알 수 있다.

• (P,C,AD)→ (T,C,AD)→ (A,C,AD)→ (P,C,AD)

이것은 차단기가 C (Closed) 상태인 경우가 지속적으로 반복하고 있음을 알 수 있는데, 그 이유는 (A,C,AD)에서 !in 사건이 손실되어서 제어기가 받지 못하였기 때문이다. 따라서 이와 같은 사실은 Gate가 Closed 되었으면 언젠가는 반드시 Open 되어야 한다는 필연성 조건을 절대 만족하지 않은 상태 순서이다. 이러한 형태의 동작을 우리는 라이브락 (Livelock)이라고 한다. 따라서 !in 사건도 손실되어서는 안된다는 것을 알 수 있다. Fig. 4의 우측하단의 !ex 사건 손실 및 !up 사건의 손실도 마찬가지로 라이브락이 빠질 수 있다. 결론적으로 이러한 사건손실로 인하여 안전성이 훼손되고 또한 필연성을 만족하지 못한다는 사실을 알 수 있다.

라이브락과는 다른 의미인 데드락 (Deadlock)은 필연성을 저해하는 또 하나의 원인이다. 데드락은 공유 자원을 접근하기 위해 서로 다른 모델이 상호 기다림으로써 상태의 전환이 멈춰버린 것, 또는 모든 모델이 무한정 기다리는 상태를 의미한다. Fig. 4의 본 해석 모델에서는 모든 모델의 잔여시간이 [∞,∞,∞]인 경우는 존재하지 않으므로 데드락은 존재하지 않는다.

4. 실험 결과

본 실험에서는 위의 RRC 모델을 DEVSim++로 구현한 후 엔진이 제공해 주는 로그파일을 분석하여서, C-DEVS로 분석한 결과와 시뮬레이션의 결과를 비교하여, C-DEVS 형식론이 시뮬레이션의 결과와 일치한다는 사실을 검증하고자 하였다. 이를 위해 먼저 Fig. 2의 RRC 모델을 DEVSim++ 코드로 구현하였고, 출력사건이 무작위로 손실되도록 출력함수에서 별도의 손실된 사건 포트에 보내서 모의하였다. Fig. 5는 이 방법으로 시뮬레이션한 DEVSim++ 엔진이 생성한 로그파일의 일부분이다. 이를 Fig. 4와 같이 분석된 C-DEVS 모델의 인식기를 통

```
D 4 [CDefaultHandler::HandleMessage()] DEVSIM_INITIAL
I 5 T_TRAVEL
I 6 G_OPEN
I 7 C_TU
D 8 [CDefaultHandler::HandleMessage()] DEVSIM_SIMULATION
I 9 [DEVSim++ Simulation Engine] SIMULATION MODE
D 10 [DEVS][STAR(300)] RRC Simulator->MyTrain
D 11 [DEVS][Y(300)] MyTrain.0x00000000(y_ap)->RRC Simulator : Empty
D 12 [DEVS][X(300)] RRC Simulator->MyController.0x00000000(x_ap) : Empty
I 13 C_AU
D 14 [DEVS][DONE(300)] MyController->RRC Simulator : tN = 303
I 15 T_APPROACH
D 16 [DEVS][DONE(300)] MyTrain->RRC Simulator : tN = 320
D 17 [DEVS][STAR(303)] RRC Simulator->MyController
D 18 [DEVS][Y(303)] MyController.0x00000000(y_dn)->RRC Simulator : Empty
D 19 [DEVS][X(303)] RRC Simulator->MyGate.0x00000000(x_dn) : Empty
I 20 G_LOWER
D 21 [DEVS][DONE(303)] MyGate->RRC Simulator : tN = 308
I 22 C_AD
D 23 [DEVS][DONE(303)] MyController->RRC Simulator : tN = Infinity
D 24 [DEVS][STAR(308)] RRC Simulator->MyGate
I 25 G_CLOSED
D 26 [DEVS][DONE(308)] MyGate->RRC Simulator : tN = Infinity
D 27 [DEVS][STAR(320)] RRC Simulator->MyTrain
D 28 [DEVS][Y(320)] MyTrain.0x00000001(y_in)->RRC Simulator : Empty
D 29 [DEVS][X(320)] RRC Simulator->MyController.0x00000001(x_in) : Empty
I 30 C_PD
D 31 [DEVS][DONE(320)] MyController->RRC Simulator : tN = Infinity
I 32 T_PASSING
D 33 [DEVS][DONE(320)] MyTrain->RRC Simulator : tN = 350
D 34 [DEVS][STAR(350)] RRC Simulator->MyTrain
D 35 [DEVS][Y(350)] MyTrain.0x00000002(y_ex)->RRC Simulator : Empty
D 36 [DEVS][X(350)] RRC Simulator->MyController.0x00000002(x_ex) : Empty
I 37 C_TD
D 38 [DEVS][DONE(350)] MyController->RRC Simulator : tN = 352
I 39 T_TRAVEL
D 40 [DEVS][DONE(350)] MyTrain->RRC Simulator : tN = 650
D 41 [DEVS][STAR(352)] RRC Simulator->MyController
D 42 [DEVS][Y(352)] MyController.0x00000001(y_up)->RRC Simulator : Empty
D 43 [DEVS][X(352)] RRC Simulator->MyGate.0x00000001(x_up) : Empty
I 44 G_RAISE
D 45 [DEVS][DONE(352)] MyGate->RRC Simulator : tN = 357
I 46 C_TU
D 47 [DEVS][DONE(352)] MyController->RRC Simulator : tN = Infinity
D 48 [DEVS][STAR(357)] RRC Simulator->MyGate
I 49 G_OPEN
D 50 [DEVS][DONE(357)] MyGate->RRC Simulator : tN = Infinity
D 51 [DEVS][STAR(650)] RRC Simulator->MyTrain
D 52 [DEVS][Y(650)] MyTrain.0x00000000(y_ap)->RRC Simulator : Empty
```

Fig. 5. DEVSim++ simulation log for RRC

Table 1. Verification via simulation (lossless case)

ST	Event			State			TA
	TR	GA	CO	TR	GA	CO	
0	-	-	-	T	O	TU	300
300	!ap	-	?ap	A	O	AU	3
303	-	?dn	!dn	A	L	AD	5
308	-	-	-	A	C	AD	12
320	!in	-	?in	P	C	PD	30
350	!ex	-	?ex	T	C	TD	2
352	-	?up	!up	T	R	TU	5
357	-	-	-	T	O	TU	293
650	!ap	-	?ap	A	O	AU	3
653	-	?dn	!dn	A	L	AD	5
658	-	-	-	A	C	AD	12
670	!in	-	?in	P	C	PD	30
700	!ex	-	?ex	T	C	TD	2
702	-	?up	!up	T	R	TU	5
707	-	-	-	T	O	TU	293
1000	!ap	-	?ap	A	O	AU	3
1003	-	?dn	!dn	A	L	AD	5
1008	-	-	-	A	C	AD	12
1020	!in	-	?in	P	C	PD	30
1050	!ex	-	?ex	T	C	TD	2
...	...	...	...	...	...	...	...

해 올바로 동작하는지를 검증하였다. Fig. 4의 천이 모델을 인식할 대상으로 하고 DEVSim++의 로그를 입력받아 사건에 의해 상태를 천이하고, 머무는 시간을 비교하는 식으로 검증하였다. 시뮬레이션 시간은 2000초 동안 동작하도록 하였다. 로그파일을 분석하여 도달한 상태와 사건, 그리고 그 상태에서 머무는 시간이 일치하는지를 측정하였다. Fig. 5는 시뮬레이션 로그 파일의 일부분이다. DEVSim++ 로그를 분석한 결과 Fig. 4의 C-DEVS 모델에서 벗어나는 경우가 없이 동일함이 검증되었다. 이것은 C-DEVS 모델이 DEVS로 기술된 모델의 동작과 동일한 의미론을 가지고 잘 정의되었음을 보여주는 것이라고 하겠다.

Table 1은 Fig. 5의 로그파일을 분석하여 시간 순으로 정리한 것으로 무손실일 경우 Fig. 2와 완전히 일치하는 것을 볼 수 있다. 표 상단에서 ST는 시뮬레이션 시각을 의미하며, TA는 복합상태의 시간전진, TR, GA, CO는 각각 기차, 차단기, 제어기를 나타낸다. '-'로 표시된 것은 주고받는 사건이 없을 경우, 즉 무사건을 나타낸다. 예를 들어 Table 1의 ST=303은 Fig. 5의 로그 라인 10~15에 해당하며 이 로그는 사건 dn이 Controller에서 Gate로 전달되었고 해당 모델의 내부 및 외부상태천이가 각각 일어

Table 2. Verification via simulation (event !dn is lost)

ST	Event			State			TA
	TR	GA	CO	TR	GA	CO	
0	-	-	-	T	O	TU	300
300	!ap	-	?ap	A	O	AU	3
303	-	(?dn)	!dn	A	O(L)	AD	17
308	-	-	-	A	O(C)	AD	-
320	!in	-	?in	P	O(C)	PD	30
350	!ex	-	?ex	T	O(C)	TD	2
352	-	[?up]	!up	T	O(R)	TU	5
...	...	...	...	...	...	...	...

났음을 나타낸다. 이러한 방법으로 분석한 결과 사건이 손실될 때에도 분석된 시각과, 사건의 송수신, 그리고 상태의 변화, 시간전진이 Fig. 5처럼 동일하게 해석되는 것을 알 수 있었다.

Table 2는 사건 !dn이 손실되었을 때의 시뮬레이션 결과이다. 시뮬레이션 시간 303초에서 제어기(CO)가 !dn 사건을 보냈으나, 차단기(GA)는 ?dn 사건을 받지 못하였음을 나타낸다. 따라서 GA의 상태는 Open의 상태에 머무르게 된다. 따라서 시간 308초에서 게이트가 닫치는 사건은 일어나지 않고 Open 상태에 머무른다. 시간 352초에 CO가 !up 명령을 내렸으나, Fig. 2의 모델에서처럼 GATE가 Open 상태에서는 ?up 명령을 받지 않기 때문에 시뮬레이션이 종료되었다. Table 2의 결과도 Fig. 4의 C-DEVS 모델에서 해석된 결과와 일치함을 알 수 있다. 많은 시간 실험을 반복한 결과 랜덤하게 손실되는 경우가 모두 발생하였으며, 이는 해석결과인 Fig. 4의 상태 및 상태천이를 모두 만족하는 것으로 나타났다.

위 실험으로 알 수 있는 것은 제안된 C-DEVS의 상호작용 규칙이 시뮬레이션 모델에서의 규칙과 완전히 적합하다는 것을 증명하고 있다. 하지만 C-DEVS 형식론에 의한 해석은 시뮬레이션과는 달리 모든 가능한 행위를 얻을 수 있다는 것이 유의해야 한다. 따라서 C-DEVS 형식론을 사용하여 얻어진 해석모델은 안전성과 필연성 등 여러 가지 시간 논리 특성을 모델 검사 기법을 통해 분석하기에 잘 정의된 적합한 모델임을 알 수 있다.

5. 결 론

본 논문은 실시간 이산사건 시스템의 논리적인 특성을 C-DEVS 형식론을 이용하여 해석하는 방법을 제안하였다. C-DEVS는 DEVS 원자모델들의 상호작용을 수학적으로 정의한 것인데 본 논문에서는 이 정의를 적용하여

C-DEVS 모델을 만드는 알고리즘을 제안하고 RRC 모델을 통해 적용사례를 보여주었으며 특히 안전성과 필연성에 대해서 정의하고 분석하였다. 또한, DEVSim++를 활용하여 RRC 모델을 구현하고 로그파일을 생성하여, 알고리즘을 적용하여 얻어진 C-DEVS 모델과 시뮬레이션 결과와 일치함을 검증하였다. 이는 C-DEVS 형식론이 원래의 DEVS 형식론에서 정의된 동작의 의미를 명확하게 정의하고 있음을 나타내는 것이다. C-DEVS 형식론은 기존과는 달리 추가적으로 사건이 손실되는 경우까지 포함하는 상호작용을 정의하고 있어서 현실적으로 통신 장애 등으로 인한 모델의 안전성 등의 특성을 분석해 볼 수 있다는 것을 알 수 있었다. 다만, 상태폭발의 문제는 피할 수 없는 한계점으로 추가 연구를 통해 완화시킬 수 있는 방법을 모색해야 할 것으로 사료된다. 실시간 제어기를 설계하는 문제와, 시간이 미정일 때 안전한 동작을 위한 시간을 정하는 문제, 그리고 DEVS 원자모델이 주어졌을 때 자동으로 C-DEVS 모델을 만들고, 해석하는 도구에 대한 계속적인 연구가 필요할 것으로 사료된다.

참 고 문 헌

1. Leveson NG, Stolzy JL. Safety Analysis Using Petri Nets. IEEE Trans. Software Engineering 1987; SE-13: 386-397.
2. Song HS, Kim TG. Application of Real-Time DEVS to Analysis of Safety-Critical Embedded Control Systems: Railroad Crossing Control Example. Simulation 2005; 81: 119-136.
3. Bengtsson J, Yi W. Timed Automata: Semantics, Algorithms and Tools. LNCS 2004; 3098: 87-124.
4. Boucheneb H, Barkaoui K. Relevant Timed Schedules/Clock Vectors for Constructing Time Petri Net Reachability Graphs. Discrete Event Dynamic Systems 2011; 21: 171-204.
5. Zeigler BP, Kim TG, Praehofer H. Theory of Modeling and Simulation, Orlando, FL: Academic 2000.
6. Lee WB, Kim TG. Ordering Method for Reducing State Space in Compositional Verification. in 1999 IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC '99), Tokyo, Japan 1999; I-806-I-811.
7. Song HS, Kim TG. Safety Analysis of Computer-controlled Real-time Systems with Message Loss Using Communicating DEVS Models. AsiaSim 2012, Part I, Communications in Computer and Information Science, Springer-Verlag Berlin Heidelberg 2012; 323: 480-489.
8. Sung CH, Koo J, Kim TG, Kim KH. Verification of Automatic PAR Control System using DEVS formalism, Journal of Korea Simulation Society 2012; 21: 1-9.
9. Saadawi H, Wainer G. Verification of Real-Time DEVS Models, Proceedings of the 2009 Spring Simulation Multi-conference, Society for Computer Simulation International 2009; Article no. 143.
10. Choi CB, Kim TG. Software Formal Verification Methodology using Aspect DEVS Verification Framework, Journal of Korea Simulation Society 2009; 18: 113-122.

송 해 상 (hssong@seowon.ac.kr)

1991 한국과학기술원 전기및전자공학과 공학석사
2000 한국과학기술원 전자전산학과 공학박사
1999～2000 고등기술연구원 연구원
2002～현재 서원대학교 컴퓨터공학과 교수

학술활동 : 한국시뮬레이션 학회 종신회원, 한국시스템공학회 이사
관심분야 : 이산사건시스템 M&S 이론 및 응용, 국방시스템공학.

김 탁 곤 (tkim@ee.kaist.ac.kr)

1988 Univ. of. Arizona, 전기및컴퓨터공학과 박사
1980～1983 부경대학교, 통신공학과, 전임강사
1987～1989 (미)아리조나 환경연구소, 연구엔지니어
1989～1991 Univ. of Kansas, 전기및컴퓨터공학과, 조교수
1991～현재 KAIST 전기및전자공학과, 교수
- 한국시뮬레이션 학회 회장 역임
- 국제시뮬레이션학회(SCS) 논문지(Simulation) Editor-In-Chief 역임
- SCS Fellow
- 모델링 시뮬레이션 기술사(미국), Who's Who in the World(Marguis 16thEdition, 1999) 등재
- 연합사, 국방부/합참, 기품원 자문위원 역임, KIDA Fellow 역임, ADD 자문위원(현)

관심분야 : 모델링/시뮬레이션 이론, 방법론 및 환경개발, 시뮬레이터 연동