ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 42, No. 7, pp. 823-833, 2015. 7
http://dx.doi.org/10.5626/JOK.2015.42.7.823

하이브리드 시스템을 명세하기 위한 ETRI CPS 모델링 언어

(An ETRI CPS Modeling Language for Specifying Hybrid Systems)

윤 상 현 † 전 인 걸 †† 김 원 태 †† 조 재 연 †† 유 준 범 †††
(Sanghyun Yoon) (In-geol Chun) (Won-Tae Kim) (Jaeyeon Jo) (Junbeom Yoo)

요 약 하이브리드 시스템은 연속 시스템과 이산 시스템으로 구성된 동적 시스템이다. 하이브리드 시스템은 자동차, 항공, 군사 방어 등 시스템을 명세 하는데 사용되고 있으며 이를 위해 다양한 모델링 언어와 지원 도구가 개발되고 사용되어 왔다. 제안되어 사용되고 있는 언어와 도구들은 목적에 따라 특정한 특징들을 갖고 있다. 한국전자 통신연구원에서 제안한 하이브리드 시스템 모델링 언어인 ECML (ETRI CPS Modeling Language)은 DEV&DESS (Differential Event and Differential Equation Specified System) 형식론을 CPS (Cyber-Physical System) 환경에 맞게 확장한 언어이며 모델링 및 시뮬레이션을 지원한다. 논문에서는 ECML을 소개하고 정형 정의를 제안한다. 또한 제안된 정의에 따라 간단한 차량 모델을 명세한 사례연구를 수행한다.

키워드: 하이브리드 시스템, ECML, DEV&DESS, CPS

Abstract Hybrid system is a dynamic system that is composed of both a continuous and discrete system, suitable for automobile, avionic and defense systems. Various modeling languages and their supporting tools have been proposed and used in the hybrid system. The languages and tools have specific characteristics for their purpose. Electronics and Telecommunications Research Institute (ETRI) proposed a hybrid system modeling language, ECML (ETRI CPS Modeling Language). ECML extends DEV&DESS (Differential Event and Differential Equation Specified System) formalism with consideration of CPS (Cyber-Physical System), which supports modeling and simulation. In this paper, we introduce ECML and suggest a formal definition. The case study specifies a simple vehicle model using the suggested formal definition.

Keywords: hybrid system, ECML, DEV&DESS, CPS

· 본 연구는 산업통산자원부와 방위사업청 주관의 민군기술협력진흥센터의 지원으로 수행되었습니다.

† 학생회원 : 건국대학교 컴퓨터공학부
pctkdgus@konkuk.ac.kr
†† 비 회 원 : 한국전자통신연구원 임베디드소프트웨어연구부 연구원(ETRI)
igchun@etri.re.kr
wtkim@etri.re.kr
jaeyeonjo@etri.re.kr
(Corresponding author임)
††† 정 회 원 : 건국대학교 컴퓨터공학부 교수
jbyoo@konkuk.ac.kr

논문접수 : 2014년 7월 21일
(Received 21 July 2014)
논문수정 : 2015년 3월 22일
(Revised 22 March 2015)
심사완료 : 2015년 4월 12일
(Accepted 12 April 2015)

정보과학회논문지 제42권 제7호(2015. 7)

1. 서 론

하이브리드 시스템은 자동차, 항공, 군사 방어 시스템 등을 명세하며 주로 시뮬레이션을 위해 사용되는 시스템으로, 연속 시스템(continuous system)과 이산 시스템(discrete system)이 혼합된 동적 시스템이다. 이산 시스템은 대체로 하이브리드 시스템의 작동 모드(operation mode)를 모델링 하는데 사용되며, 연속 시스템은 하이브리드 시스템 내부간의, 혹은 외부와의 물리적인 상호작용을 모델링 하는데 사용된다.

하이브리드 시스템을 명세하기 위한 언어 및 지원도구는 크게 시뮬레이션과 정형검증 특히, 모델 체킹을 목표로 하는 두 가지로 나눌 수 있다. 하이브리드 시스템을 시뮬레이션 하기 위한 언어 및 도구에는 Simulink[1], CHARON[2] 등이 있으며 정형검증을 목적으로 하는 언어와 도구에는 timed automata[3], UPPAAL[4], d/dt[5], 그리고 hybrid automata[6] 기반의 언어들인 linear hybrid automata[7], I/O hybrid automata[8]가 있으며 hybrid automata 계열 언어를 지원하는 도구에는 HyTech[9], PHAver[10], SpaceEx[11] 등이 사용되고 있다.

ECML(ETRI CPS Modeling Language)[12]는 한국전자통신 연구원(Electronics and Telecommunications Research Institute, ETRI)에서 제안한 하이브리드 시스템 모델링 언어이며 시뮬레이션을 목적으로 한다. ECML은 모듈화 및 계층성을 지원하며 DEV&DESS (Discrete Event and Differential Equation Specified System) 형식론(formalism)[13]을 기반으로 CPS(Cyber Physical System)[14]에 맞게 확장한 언어이다. ECML을 이용한 모델링 및 시뮬레이션이 수행되고 있으며 모델링 도구인 EcoPOD (ETRI CPS Open Developer)와 시뮬레이터인 EcoSIM (ETRI CPS Simulator)가 이를 지원하고 있다.

하이브리드 시스템이 사용되는 환경을 고려해 보았을 때, ECML은 안전 필수 시스템(safety-critical system)을 명세 하는데 사용될 수 있으며, 이에 따라 정형 검증을 통해 안전성(safety), 도달 가능성(reachability) 등을 검증해 볼 필요가 있다. 그러나 ECML에 대한 명확한 정형 정의가 부족하여 정형 검증을 수행하기에 어려울 수 있으며, 다양한 이해 당사자간의 의사소통에 문제가 발생할 수 있었다. 논문에서는 이를 해결하기 위해 ECML을 위한 정형 정의를 제안한다. 또한, ECML의 기본적인 동작방식은 DEV&DESS로 명세한 시스템과 유사하지만 CPS 환경에 맞게 변형되거나 추가된 부분들이 있으므로 이러한 차이점에 대해 설명하고 ECML로 정의한 차량 모델을 정형 정의에 따라 정의한 사례연구를 보인다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 ECML의 기반이론인 DEV&DESS를 개요를 설명 한다. DEV&DESS의 기본적인 시멘틱에 대한 설명과 간단한 예제를 보인다. 3장에서는 ECML의 정형 정의를 보이고 DEV&DESS와의 차이점을 이야기 한다. 4장에서는 ECML을 이용하여 차량 모델을 명세한 사례연구를 수행한 결과를 보인다. 5장에서 관련 연구와 향후 연구 계획을 소개하고 6장에서 결론을 내린다.

2. DEV&DESS

DEV&DESS는 하이브리드 시스템을 시뮬레이션 하기 위한 형식론으로, DEVS(Discrete Event System Specification)와 DESS(Differential Equation System Specification)이 혼합되어 있다. DEV&DESS는 그림 1과 같이 이산 사건(discrete event, X^{discr})과 연속 값 입력(continuous value input, X^{cont})을 입력으로 받으며, 입력에 따라 연속 상태(continuous state, S^{cont}) 및 하이브리드 오토마타의 location과 유사한 역할을 하는 이산 상태(discrete state, S^{discr})를 변화시키며, 이에 따른 이산 사건 출력(discrete event output, Y^{discr})과 연속 값 출력(continuous value output, Y^{cont})을 내보낸다.

DEV&DESS는 이산 사건이나, 혹은 연속 값 입력이나 연속 상태가 조건을 만족하는 (내부) 상태 사건의 발생에 따라 세가지 동작 방식을 갖는다. 간략하게 소개하면 다음과 같다. 첫째, 일정 시간 간격 동안 아무런 사건이 발생하지 않았을 때는 이산 상태가 변화하지 않고, 시간이 증가하며 그에 따라 연속 상태가 변화한다. 변화된 연속 상태 값에 따라 연속 값 출력이 정의되어 나간다. 둘째, 내부 상태 사건이 일정 시간 간격 내에 발생하면 이산 상태가 바뀌고 이산 사건 출력이 발생한다. 내부 상태 사건이 발생한 시간까지 변화한 연속 상태에 따라 연속 값 출력도 정의되어 나간다. 셋째, 이산 사건이 일정 시간 간격 내에 발생하면 이산 상태가 변화하고, 사건이 발생한 시간까지 변화한 연속 상태에 의해 연속 값 출력이 정의되어 나간다. 여기에서 내부 상태 사건은 DEV&DESS 모델 내부의 상태가 특정 조건 함수를 만족하였을 때를, 이산 사건은 외부의 입력으로 들어오는 이산 사건(X^{discr})이 발생하였을 때 대응하는 것을 의미하는 차이가 있다.

DEV&DESS의 여러 개의 단일한 DEV&DESS 모델 간의 포트를 이용한 데이터 전달이 가능하다. 여러 개의 단일 DEV&DESS 모델이 연결된 전체 모델을 혼합 모델(coupled model)이라고 하며 DEV&DESS로 표현된 전체 모델을 표현할 때 사용된다. 이는 다음 장에서 설명할 ECML의 구조모델과 유사하다.

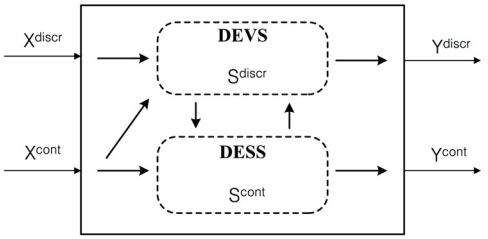

그림 1 DEV&DESS 단일 모델[13]
Fig. 1 DEV&DESS atomic model

3. ECML

3.1 ECML 구조 모델과 행위 모델

ECML은 DEV&DESS 형식론을 기반으로 하여CPS에 맞게 확장한 언어이다. ECML 모델은 구조 모델(*SM*, Structural Model)과 행위 모델(*BM*, Behavioral Model)로 구성되어 있다. ECML 모델은 하나 이상의 구조 모델로 구성되며, 구조 모델간의 계층적인 관계를 가지고 있어 전체 ECML 모델을 캡슐화(encapsulation) 및 재사용을 지원할 수 있게 되어 있다. 그림 2와 같이 구조 모델은 여러 개의 행위 모델로 구성되어 있고 각각의 행위 모델은 포트를 이용하여 행위 모델간에 혹은 행위모델과 구조모델간 변수의 값을 주고 받을 수 있다. 하얀 정사각형이 입력 포트를 나타내며 검은 정사각형은 출력 포트를 나타낸다.

행위 모델은 구조 모델의 내부적인 상태 변화와 상태 변화에 따른 다양한 변수의 변화를 표현한다. 그림 3은 *'Barrel filler'* 모델을 ECML의 행위 모델로 표현한 것이다. *'Barrel filler'* 모델은 두 개의 입력과 출력을 갖는다. 밸브(*Valve*)가 열려있는 동안 배럴에 물의 유입량(*inflow*)만큼 물을 채우고 배럴의 물의 높이(*Level*)가 10이 되면 배럴을 내보내고(*Barrel*) 새로운 배럴에 물을 채우는 모델이다. 현재 배럴의 물의 높이는 *'LevelOut'*

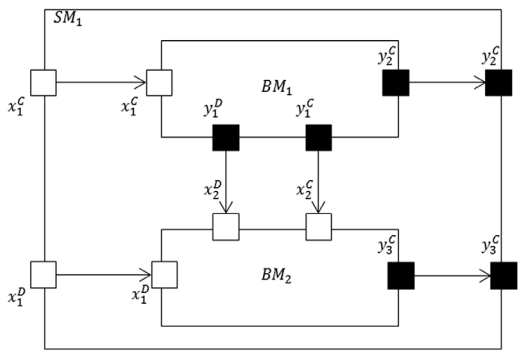

그림 2 ECML 구조 모델
Fig. 2 A structural model of ECML

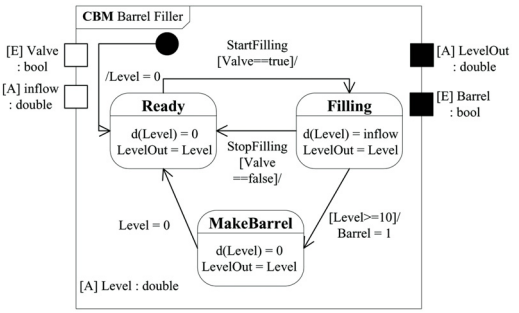

그림 3 ECML 행위 모델로 표현한 Barrel Filler 모델
Fig. 3 A Barrel Filler model described in behavioral model in ECML

을 통해 계속해서 출력된다. *'Ready'*, *'Filling'*, *'MakeBarrel'*의 세 phase으로 구성되며 각 phase에서 연속 상태인 *'Level'*의 변화율이 정의되고 (*d(Level)=0*) 출력인 *'Level-Out'*에 연속 상태 값인 *'Level'*의 값을 할당(*LevelOut=Level*)해주고 있다. Phase에서 일정 조건을 만족하면 전이가 발생하여 모델의 현재 상태가 다른 phase로 정의될 수 있는데, 예를 들어 이산 사건인 *'Valve'*가 *true*가 되면 *'Ready'*에서 전이가 발생하여 *'Filling'*으로 다음 phase가 정의되거나 *'Valve'*의 값이 *false*일 때는 반대로 *'Filling'*에서 *'Ready'*로 전이가 발생한다.

각각의 입출력 포트에는 *'[A], [E]'*와 같은 갱신 명세자(update specifier)가 있어 해당 포트가 어떤 형의 변수인지를 표현한다. 세 종류의 갱신 명세자가 있으며 '[A]'는 analog, 연속 값 변수를, '[E]'는 event, 이산 사건을, '[D]'는 이산 값(discrete value) 변수를 의미한다. 연속 값 변수 및 이산 사건 변수는 DEV&DESS 변수들과 같은 타입의 변수이다. 이산 값 변수는 DEV&DESS에는 없는 형태의 변수인데 그림 4와 같이 특정 시간에 사건이 하고 값이 '0'으로 초기화되는 이산 사건 변수와는 다르게 이산 값 변수는 한번 정의된 값이 다시 정의되기 전에는 계속해서 유지된다는 차이점이 있어 다양한 모델이 연결되어 값을 주고 받는 CPS 시스템에서 유용하게 사용될 수 있다. 또한, DEV&DESS의 이산 상태(S^{discr})는 phase와 유사한 기능을 의미하지만 ECML에서의 이산 상태는 이산 값 변수의 값이 유지되고 있는 것으로 해석되며 이에 따른 전이가 발생할 수도 있다.

추가적으로 시뮬레이션을 위해 모델 전체의 외부의 센서 입출력 등을 계산해주는 환경모델이 있다. 현재 환경 모델은 ECML의 구조 모델 및 행위 모델을 기초로 시뮬레이션과 관련된 복잡한 계산을 위해 C 함수 등을 포함 하고 있으므로 논문에서 정형 정의 및 예제의 대상으로 하지 않고 모델링에 관련된 구조 모델과 행위

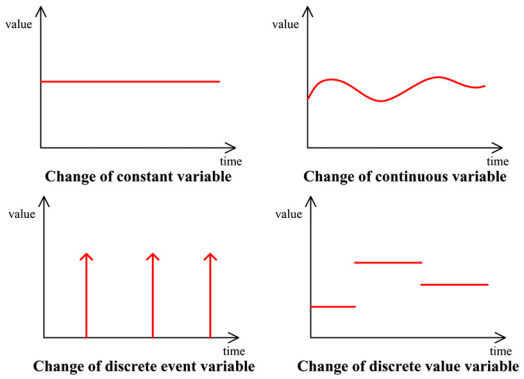

그림 4 변수의 종류에 따른 변수 값의 변화

Fig. 4 Change of variable value according to variable types

모델만을 언급한다. 이 후 환경모델은 행위 모델과 구조 모델의 정의만을 이용하여 모델링 할 계획이다.

3.2 ECML의 정형 정의

ECML 모델은 DEV&DESS를 기반으로 하고 있기 때문에 DEV&DESS의 세 가지 동작방식과 기본적으로 유사하게 동작한다. 그러나 이산 값 변수를 추가적으로 사용하여 연속 값과 같이 이산 값이 계속 해서 출력으로 나갈 수 있는 점이 다르다. 연속 상태 및 이산 상태는 각각 지역 변수의 값을 저장하는 용도로 사용되어 DEV&DESS의 이산 상태의 역할은 phase가 담당하는 것이 달라 모델은 initial phase에서부터 실행하게 된다. 그리고 외부의 입력 중 이산 사건이 발생 했을 때만을 전이의 조건으로 사용한 DEV&DESS와 다르게 ECML은 이산 사건뿐만 아니라 연속 값, 이산 값 변수가 입력으로 들어왔을 때 전이의 조건으로 사용할 수 있다. 따라서 ECML의 정형 정의는 외부 사건 전이 함수에 이를 고려하도록 하였다. 내부 상태 사건도 연속 상태뿐만 아니라 이산상태도 고려하도록 하였다. ECML의 행위 모델에 대한 정형 정의는 DEV&DESS의 정의를 확장한 형태로 정의되어 있으며 다음과 같이 표현할 수 있다.

$$BM = \langle X, Y, S, Cond^E, Trans^E, Trans^S, Cond^S, Rate, Out^C, Out^D, Out^E, Out^S \rangle$$

$X = X^C \times X^D \times X^E$ 는 입력들의 집합이다.

$X^C = \{(x_1^C, x_2^C, \cdots) \mid x_1^C \in X_1^C, x_2^C \in X_2^C, \cdots\}$ 는 입력 변수들 x_i^C 를 포함한 연속 값 입력들의 구조화된 집합,

$X^D = \{(x_1^D, x_2^D, \cdots) \mid x_1^D \in X_1^D, x_2^D \in X_2^D, \cdots\}$ 는 입력 변수들 x_i^D 를 포함한 연속 값 입력들의 구조화된 집합,

X^E 는 이산 사건 입력들의 집합,

$Y = Y^C \times Y^D \times Y^E$ 는 출력들의 집합이다.

$Y^C = \{(y_1^C, y_2^C, \cdots) \mid y_1^C \in Y_1^C, y_2^C \in Y_2^C, \cdots\}$ 는 출력 변수들 y_i^C 를 포함한 연속 값 출력들의 구조화된 집합,

$Y^D = \{(y_1^D, y_2^D, \cdots) \mid y_1^D \in Y_1^D, y_2^D \in Y_2^D, \cdots\}$ 는 출력 변수들 y_i^C 를 포함한 연속 값 출력들의 구조화된 집합,

Y^E 는 이산 사건 출력들의 집합,

$S = P \times S^D \times S^C$ 는 상태들의 집합이다. 상(phase) {initial phase) $\in P$, 이산 상태 S^D, 연속 상태 S^C 의 카테시안 곱,

$Cond^E : S \times X \rightarrow Bool$ 는 외부 사건들의 실행에 영향을 미치는 외부 사건 전이 조건,

$Trans^E : S \times X \rightarrow S$ 는 외부 사건 전이 함수,

$Out^E : S \times X \rightarrow Y$ 는 외부 사건 전이들에 대한 출력 함수,

$Cond^S : S \times X^C \times X^D \rightarrow Bool$ 는 내부 상태 사건들의 실행에 영향을 미치는 상태 전이 조건,

$Trans^S : S \times X^C \times X^D \rightarrow S$ 는 내부 상태 전이 함수,

$Out^S : S \times X^C \times X^D \rightarrow Y$ 는 내부 상태 전이에 대한 출력 함수,

$Rate : S \times X^C \times X^D \rightarrow S^C$ 는 변화 함수의 비율.

$Out^C : S \times X^C \times X^D \rightarrow Y^C$ 는 연속 값 출력 함수,

$Out^D : P \times S^D \times X^D \rightarrow Y^D$ 는 이산 값 출력 함수,

일정시간 간격 $<t_1, t_2>$ ($[t_1, t_2]$ 또는 (t_1, t_2)에 따른 ECML 행위 모델에 대한 비정형적 시멘틱은 다음과 같다.

1. **일정 시간 간격 $<t_1, t_2>$ 안에 사건이 발생하지 않을 때:** 연속 상태 S^C 만 변화한다. 연속 상태의 시간 간격의 끝의 값은 초기값에 시간 간격 동안의 변화함수의 비율 $Rate(s(t), x^C(t), x^D(t))$ ($t = <t_1, t_2>$)의 적분을 더한 값이 된다. 모델의 연속 행위(behavior)는 $Rate(s(t), x^C(t), x^D(t))$와 연속 값 출력 함수 $Out^C(s(t), x^C(t), x^D(t))$에 의해 결정되고, 이산 값 출력은 이산 값 출력 함수 $Out^D(p(t), s^D(t), x^D(t))$가 생성한다.
2. **내부 상태 사건이 일정 시간간격 $<t_1, t_2>$ 내의 시각 t 에 발생할 때:** 전이가 발생하는 시각의 연속 상태의 값은 시간 간격내의 초기값에 t 까지의 변화함수의 비율 $Rate(s(t'), x^C(t'), x^D(t'))$ ($t' = <t_1, t]$)의 적분을 더한 값이 된다. 또한, 하이브리드 출력도 시각 t 까지 생성된다. 시각 t 일 때, 상태 전이 조건 함수 $Cond^S(s(t), x^C(t), x^D(t))$가 *true* 가 되어 내부 상태 사건이 발생한다. 이 때, 내부 상태 전이 함수 $Trans^S(s(t), x^C(t), x^D(t))$가 실행되어 새로운 상태를 정의한다. 내부 상태 전이에 대한 이산 사건 출력 함수 $Out^S(s(t), x^C(t), x^D(t))$가 호출되어 시각 t 일 때 출력을 생성한다.
3. **외부 사건이 일정 시간간격 $<t_1, t_2>$ 내의 시각 t 에 발생할 때:** 전이가 발생하는 시각의 연속 상태의 값은 시간 간격내의 초기값에 t 까지 변화함수의 비율 $Rate(s(t'), x^C(t'), x^D(t'))$ ($t' = <t_1, t]$)의 적분을 더한 값이 된다. 또한, 연속 출력도 시각 t 까지 생성된다. 시각 t 일 때, 외부 사건 조건 함수 $Cond^E(s(t), x(t))$가 *true*가 되어 외부 사건이 발생한다. 이 때, 외부 사건 전

이 함수 $Tran^E(s(t), x(t))$가 실행되어 새로운 상태를 정의한다. 외부 사건 전이에 대한 이산 사건 출력 함수 $Out_E(s(t), x(t))$가 호출되어 시각 t 일 때 출력을 생성한다.

여러 개의 행위 모델이 합쳐진 형태로 볼 수 있는 구조 모델의 정의는 DEV&DESS의 네트워크를 이용한 결합 모델의 정의 즉, *'Influencer/Influencee'* 관계를 이용한 정의와 큰 차이가 없다. 여기에서 행위 모델 또는 구조 모델의 influencer는 해당 모델의 입력에 변수의 값을 전달해 주는 모델을 이야기하며, 값을 받는 모델은 influencee가 된다. 구조모델에 대한 정의는 다음과 같이 표현할 수 있다.

$$SM = \langle X_S, Y_S, B, \{M_b\}, \{I_b\}, \{Z_b\}, Select \rangle$$

$X_S = X_S^C \times X_S^D \times X_S^E$ 는 구조모델의 입력들의 집합,

$Y_S = Y_S^C \times Y_S^D \times Y_S^E$ 는 구조모델의 출력들의 집합,

B 는 구조모델에 포함된 행위모델들의 참조(reference)의 집합,

$Select$ 는 실행될 모델 선택 함수,

각 $b \in B$ 에 대해, M_b 는 행위 모델,

각 $b \in B \cup \{S_R\}$에 대해,

I_b 는 b 의 *influencer* 집합: $I_b \subseteq B \cup \{S_R\}$, $b \notin I_b$

$\{R_s\}$는 SM의 참조(reference)

각 $i \in I_b$ 에 대해,

$Z_b(\{R_S\}, X_s, Y_S, Y_i, X_b)$는 $i-to-b$ 출력 변환 함수,

$i = R_S$ 이면, $Z_b : X_S \rightarrow X_b$, $Select = M_b$

$b = R_S$ 이면, $Z_b : Y_i \rightarrow Y_S$,

$b \neq R_S$ 이고 $i \neq R_S$ 이면, $Z_{i,b} : Y_i \rightarrow X_b$,

$Select = M_b$.

Z_b 가 여러 개가 있을 경우,

$||Z_{bi} = (Z_{b1}||Z_{b2}||\cdots||Z_{bn})$)는 모든 $i-to-b$ 출력 변환 함수의 parallel composition이다. i 에 대한 모든 $i-to-b$ 출력 변환 함수의 집합 Z_{bi} 에 대해 $\{Z_{b1} \in Z_{bi}, Z_{b2} \in Z_{bi}, \cdots, Z_{bn} \in Z_{bi}\}$

구조모델의 정의에서 B는 구조모델에 포함된 행위모델의 이름, 아이디 등으로 표현되는 참조부호의 집합을 이야기하는 반면, $\{M_b\}$는 해당 행위 모델의 정의를 포함한 집합이라는 점에서 차이가 있다. 출력 변환 함수인 Z_b는 구조 모델에 포함된 모든 행위 모델 또는 해당 구조 모델에 대해 입력 포트에 값을 주는 influencer, I_b를 이용하여 모델간의 연결 관계를 파악하여 실제 연결된 포트간의 값 할당은 $\{M_b\}$에 포함되어 있는 포트 정보를 이용하여 출력 변환 함수인 Z_b에 따라는 정의를 하게 된다. 하나의 출력(influencer)이 다른 여러 모델의 여러 입력과 연결되어 동시에 실행 할 수 있는 컴포넌트가 여러 개 있을 경우 이를 병렬적으로 처리하고 (parallel composition) Z_b의 대상이 행위모델일 경우 다음에 수행될 모델로 선택한다.

4. 사례연구

차량 모델을 대상으로 3장의 정형정의에 따라 모델을 정의하는 사례연구를 수행한다. 차량 모델은 외부에서 차량을 목표 위치까지 이동시키기 위해 차량의 현재 위치와 목표 위치를 비교하여 목표 위치까지 직선으로 진행 방향을 정하고 엔진의 출력을 변화시키는 모델이다. 그림 5는 ECML모델링 및 시뮬레이션 환경인 EcoPOD에서 명세한 차량의 진행 방향과 그에 따른 차량의 위치 변화를 보여주고 있는데, 차량은 직선으로 이동하지만 진행 방향(*heading*)에 따라서 엔진의 출력이 같더라도 실제 x, y 좌표의 시간당 변화량은 달라지게 된다. 실제 차량 모델은 ECML의 환경모델을 이용하여 sin, cos 함수를 적용해 차량의 현재 좌표를 계산하는 과정이 있으나 논문에서 보일 차량 모델은 환경 모델을 생략하였다. 또한 진행 방향은 본래 360°어느 방향으로든지 선택할 수 있게 되어 있지만 45°, 135°, 225°, 315°로 한정하는 등 단순화 시킨 모델이다.

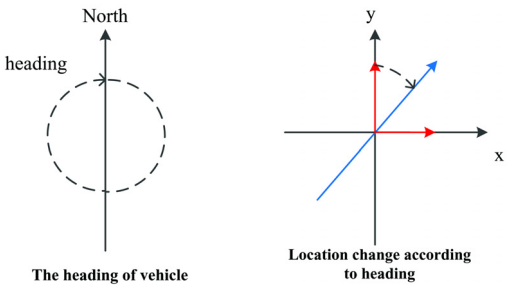

그림 5 차량의 진행방향에 따른 차량의 위치 변화

Fig. 5 Location change of a vehicle according to heading of the vehicle

그림 6은 EcoPOD에서 명세한 차량 모델을 보여주고 있다. CSM(CPS Structural Model), CBM(CPS Behavioral Model)은 각각 구조 모델과 행위 모델을 이야기하며 그림과 같이 차량 모델은 하나의 구조 모델과 그 내부를 구성하는 *'control'*, *'thrustcontroller'*, *'Engine'*의 세 개의 행위 모델로 구성되어 있다. 차량 모델의 외부 입력과 출력은 다음과 같다.

- *locationx* : 현재 차량의 x 좌표를 나타내는 입력
- *locationy* : 현재 차량의 y 좌표를 나타내는 입력
- *order* : 차량의 동작 여부를 결정하는 사건
- *destinationx* : 차량의 목표위치의 x 좌표를 나타내는 입력
- *destinationy* : 차량의 목표위치의 y 좌표를 나타내는 입력
- *outheading* : 차량의 진행 방향을 나타내는 출력

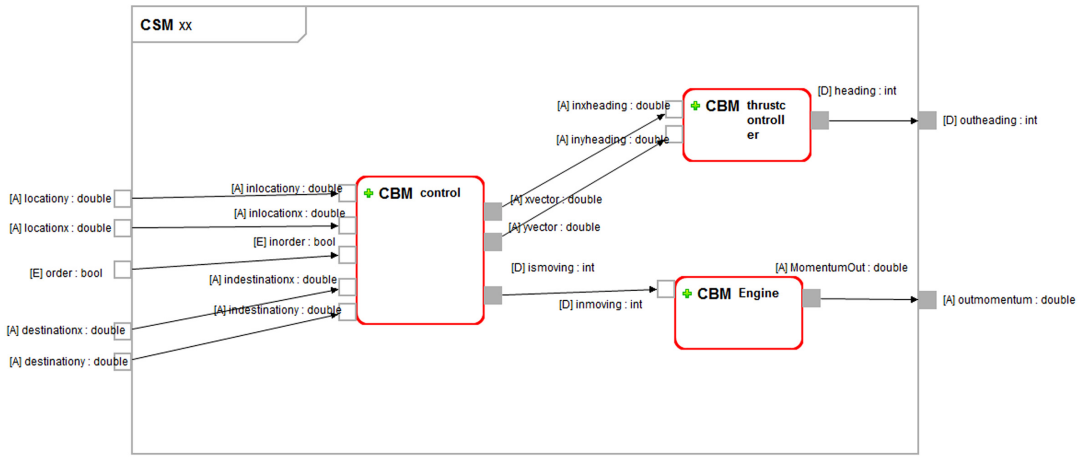

그림 6 EcoPOD 환경에서 ECML의 구조 모델로 명세한 차량 모델

Fig. 6 A vehicle model specified with ECML structural model in EcoPOD environment

• *outmomentum* : 엔진의 출력에 의한 가속도의 크기를 나타내는 출력

현재 위치를 나타내는 좌표는 위에서 언급한 것처럼 환경 모델에서 계산하여 입력으로 넣어주게 되는데 이를 계산하기 위해서 가속도를 나타내는 차량 모델은 *'outmomentum'*과 *'outheading'*을 출력으로 보내며 환경모델이 이를 받아 자동차의 계산된 현재 위치와 진행 방향 등을 결정한다. *'outheading'*의 경우 목표 위치가 변화하거나 자동차가 목표 위치를 지나치지 않는다면 결정된 값이 변화하지 않으므로 이산 값으로 정의하였다.

구조 모델에 대한 정의는 다음과 같이 표현할 수 있으며, $\mathbb{R}$은 실수 집합을, $\mathbb{N}$은 자연수 집합을 의미한다.

$$xx = \langle X_{SM}, Y_{SM}, B, \{M_b\}, \{I_b\}, \{Z_b\} \rangle$$

$$X_S^C = \{\, locationy, locationx, destinationx, destinationy \mid locationx \in \mathbb{R}, locationx \in \mathbb{R}, destinationx \in \mathbb{R},\ destinationy \in \mathbb{R}\}$$

$$X_S^E = \{order \mid order \in \{true, false\}\}$$

$$Y_S^C = \{outmomentum \mid outmomentum \in \mathbb{R}\}$$

$$\{R_S\}=xx$$

$$B = \{control, thrustcontroller, Engine\}$$

$$\{I_b\} = \{I_{control}, I_{thrustcontroller}, I_{Engine}, I_{xx}\}$$

$$I_{control} = \{xx\}$$

$$I_{thrustcontroller} = \{control\}$$

$$I_{Engine} = \{control\}$$

$$I_{xx} = \{thrustcontroller, Engine\}$$

$$\{Z_b\} = \{Z_{control}, Z_{thrustcontroller}, Z_{Engine}, Z_{xx}\}$$

$$Z_{control}(locationy, locationx, order, destinationx, destinationy, inlocationy, inlocationx, inorder, indestinationx, indestinationy)$$

$$inlocationy := locationy$$

$$inlocationx := locationx$$

$$inorder := order$$

$$indestinationx := destinationx$$

$$indestinationy := destinationy$$

$$Z_{thrustcontroller}(xvecotr, yvector, ismoving, inxheading, inyheading)$$

$$inxheading := xvector$$

$$inyheading := yector$$

$$Z_{Engine}(xvector, yvector, ismoving, inmoving)$$

$$inmoving := ismoving$$

$$Z_{xx}(heading, MomentumOut, outheading, outmomentum)$$

$$outheading := heading$$

$$outmomentum := MomentumOut$$

그림 7에서 보여주고 있는 행위 모델 *'control'*의 입력들은 차량 모델의 외부 입력을 그대로 전달 받은 것들이다. *'standby'*와 *'moving'*의 두 phase로 구성이 되어 있으며, 차량 동작 명령인 *'inorder'*가 *true*가 되면 *'moving'* phase에서 계속 머무르면서 차량의 현재 위치와 목표위치의 차이를 계산하여 *'xvector'*와 *'yvector'*의 두 출력에 할당해 준다. 또 하나의 출력인 *'ismoving'*은 단순히 *'inorder'*의 값을 다른 행위 모델에 전달하기 위한 용도로 사용된다. *'standby'*에서 *'ismoving'*으로 전이될 때 전이의 조건 앞에 붙는 *'order'*나 *'moving'*에서 phase *'standby'*로 가는 전이의 *'standby'*는 label인데 이는 hybrid automata 등에서 사용하는, 같은 label이 붙은 전이를 동시에 실행되게 하는 synchronization label이 아니라 단순히 모델링 하는 사람의 편의를 위해 붙일 수 있는 것이므로 정형 정의에서는 고려하지 않는다. Phase에 연속 변수의 변화율이 명시적으로 표기되어 있지 않은데, 이는 *'control'*은 연속 상태 등을 따로 갖지 않아 연속 변수는 입력으로 들어오는 위치를 나타내는 입력 변수들뿐이기 때문이며 입력으로 들어오는 연속 변

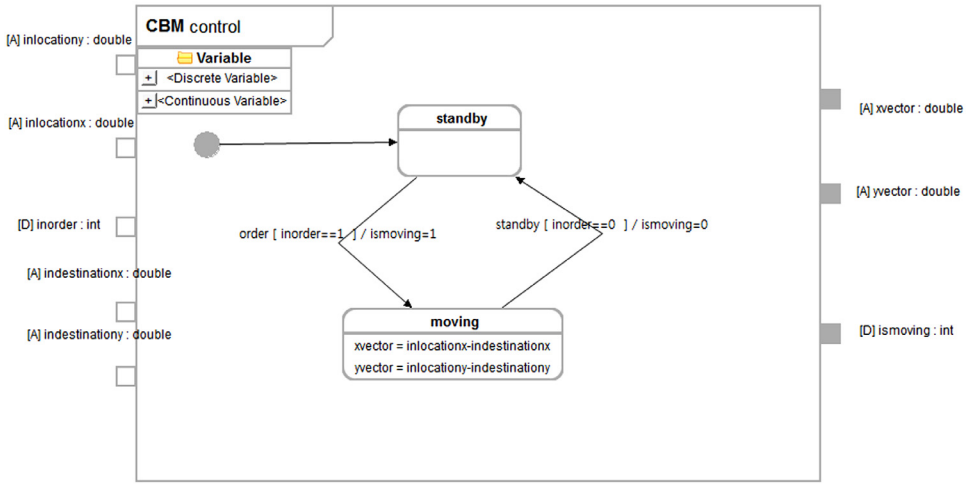

그림 7 ECML 행위 모델로 명세한 차량 모델의 일부(*control*)

Fig. 7 A part of vehicle model specified in ECML behavioral model (*control*)

수들은 전달 받아 그대로 사용하기 때문에 변화율을 지정하지 않는다. 이는 암시적으로 변화율이 '0'으로 지정된 것과 같다. 행위 모델 *'control'*의 정의는 다음과 같다.

$$control = \langle X, Y, S, Cond^E, Trans^E, Out^C, Out^E \rangle$$

$$X^C = \{inlocationx, inlocationy, indestinationx, indestinationy \mid inlocationx \in \mathbb{R}, inlocationy \in \mathbb{R}, indestinationx \in \mathbb{R}, indestinationy \in \mathbb{R}\}$$

$$X^E = \{inorder \mid inorder \in \{0,1\}\}$$

$$Y^C = \{xvector, yvector \mid xvector \in \mathbb{R}, yvector \in \mathbb{R}\}$$

$$Y^D = \{ismoving \mid ismoving \in \mathbb{N}\}$$

$$S = P$$

$$P \in \{standby, moving\}$$

$$Cond^E(phase, inorder, inlocationx, inlocationy, indestinationx, indestinationy)$$

$$phase = standby, inoder = 1$$

$$phase = moving, inoder = 0$$

$$Trans^E(phase, inorder, inlocationx, inlocationy, indestinationx, indestinationy)$$

$$if\ (phase = standby, inorder = 1)$$

$$phase \coloneqq moving$$

$$else\ if\ (phase = moving, inoder = 0)$$

$$phase \coloneqq standby$$

$$Out^E(phase, inorder,\ inlocationx, inlocationy, indestinationx,\ indestination)$$

$$if\ (phase = standby, inorder = 1)$$

$$ismoving \coloneqq 1$$

$$else\ if\ (phase = moving, inoder = 0)$$

$$ismoving \coloneqq 0$$

$$Out^C(phase, inorder,\ inlocationx, inlocationy, indestinationx,\ indestination)$$

$$if(phase = moving)$$

$$xvector \coloneqq inlocationx - indestinationx$$

$$yvector \coloneqq inlocationy - indestinationy$$

행위 모델 *'thrustcontroller'*는 행위 모델 *'control'*에서 전달해준 차량의 현재 위치와 목표 위치의 차이(*inxheading*, *inyheading*)를 입력으로 받아 차량의 진행 방향을 정해 출력 *'heading'*으로 내보내 주는 모델이다. 그림 8과 같이 현재 차량의 진행 방향을 나타내는 네 개의 phase로 구성되어 있으며 각각의 phase로 이동하는 전이에서 정해진 각도로 차량의 진행 방향을 정하도록 모델링 되어 있다. *'thrustcontroller'*에 대한 정의는 다음과 같다.

$$thrustcontroller = \langle X, Y, S, Cond^E, Trans^E, Out^E \rangle$$

$$X^C = \{inxheading, inyheading \mid inxheading \in \mathbb{R}, inyheading \in \mathbb{R}\}$$

$$Y^D = \{heading \mid heading \in \mathbb{N}\}$$

$$S = P$$

$$P = \{northenwest, northerneast, southernwest, southerneast\}$$

$$Cond^E(phase, inxheading, inyheading)$$

$$phase = northernwest, inxheading > 0$$

$$phase = northerneast, inyheading < 0$$

$$phase = southerneast, inxheading < 0$$

$$phase = southernwest, inyheading > 0$$

$$Trans^E(phase, inxheading, inyheading)$$

$$if(phase = northernwest, inxheading > 0)$$

$$phase \coloneqq northerneast$$

$$else\ if(phase = northerneast, inyheading < 0)$$

$$phase \coloneqq southerneast$$

$$else\ if(phase = southerneast, inxheading < 0)$$

$$phase \coloneqq southernwest$$

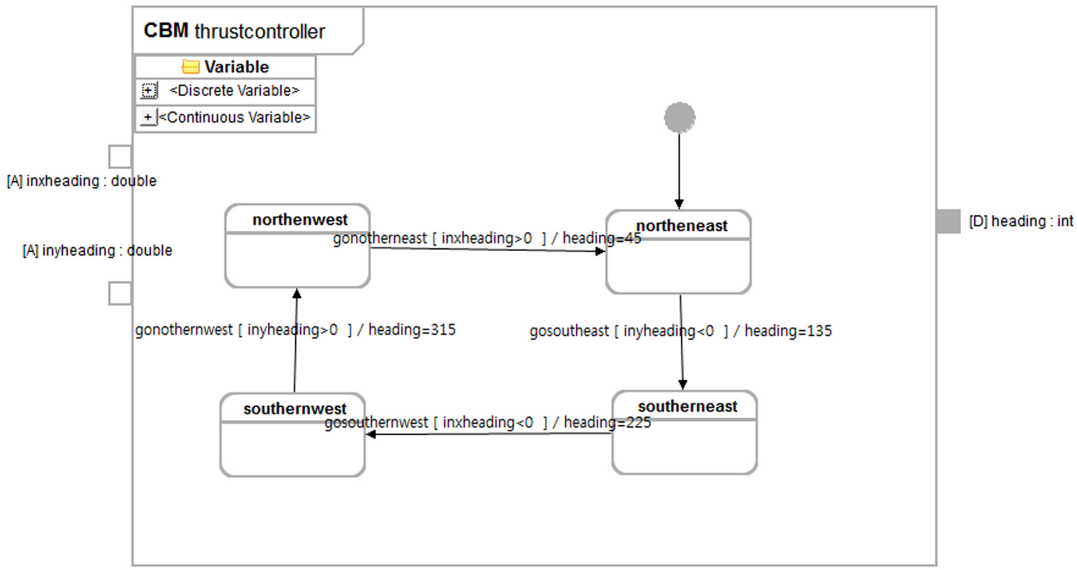

그림 8 ECML 행위 모델로 명세한 차량 모델의 일부(*thrustcontroller*)
Fig. 8 A part of vehicle model specified in ECML behavioral model (*thrustcontroller*)

$else\ if\ (phase = southernwest, inyheading > 0)$
$phase \coloneqq northernwest$
$Out^E(phase, inxheading, inyheading)$
$if\ (phase = northernwest, inxheading > 0)$
$heading \coloneqq 45$
$else\ if\ (phase = northerneast, inyheading < 0)$
$heading \coloneqq 135$
$else\ if\ (phase = southerneast, inxheading < 0)$
$phase \coloneqq 225$
$else\ if\ (phase = southernwest, inyheading > 0)$
$phase \coloneqq 315$

행위 모델 '*Engine*'은 엔진의 출력을 조절하여 차량의 가속도를 변화시키는 모델이다. 그림 9에서 보여주는 것처럼, 초기 phase인 '*Standby*'에 머무르다가 차량의 동작 명령인 '*inmoving*'의 값이 1이 되면 가속도를 증가시키는 '*Accelating*'으로 넘어가고 가속도가 60이 되는 순간 '*maintaining*'에서 현재 속도를 계속 유지시키다가 동작명령이 취소되면 '*Stopping*'에서 가속도를 감소시킨다. 그리고 모든 phase에서 현재 가속도의 크기를 출력으로 내보낸다. 이를 표현하기 위해 연속 상태인 '*Momentum*'을 사용하였고 '*Momentum*'의 변화율을 각 phase에서 정의하였다. '*Engine*'에 대한 정의는 다음과 같다.

$$Engine = \langle X, Y, S, Cond^E, Trans^E, Cond^S, Trans^S, Rate, Out^C \rangle$$

$X^D = \{inmoving \mid inmoving \in \mathbb{N}\}$
$Y^C = \{MomentumOut \mid MomentumOut \in \mathbb{R}\}$
$S = P \times S^C$
$P = \{Standby, Accelating, maintaining, Stopping\}$
$S^C = \{Momentum \mid Momentum \in \mathbb{R}\}$
$Cond^E(phase, Momentum, inmoving)$
$phase = Standby, inmoving = 1$
$phase = maintaining, inmoving = 0$
$Trans^E(phase, Momentum, inmoving)$
$if\ (phase = Standby, inmoving = 1)$
$phase \coloneqq Accelating$
$else\ if\ (phase = maintaining, inmoving = 0)$
$phase \coloneqq Stopping$
$Cond^S(phase, Momentum, inmoving)$
$phase = Accelating, Momentum = 60$
$phase = Stopping, Momentum = 0$
$Trans^S(phase, Momentum, inmoving)$
$if\ (phase = Accelating, Momentum = 60)$
$phase \coloneqq maintaining$
$else\ if\ (phase = Stopping, Momentum = 0)$
$phase \coloneqq Standby$
$Rate(phase, Momentum, inmoving)$
$if\ (phase = Standby)$
$dMomentum/dt = 0$
$else\ if\ (phase = Accelating)$
$dMomentum/dt = 1$
$else\ if\ (phase = maintaining)$
$dMomentum/dt = 0$
$else\ if\ (phase = Stopping)$
$dMomentum/dt = -1$
$Out^C(phase, Momentum, inmoving)$
$MomentumOut \coloneqq Momentum$

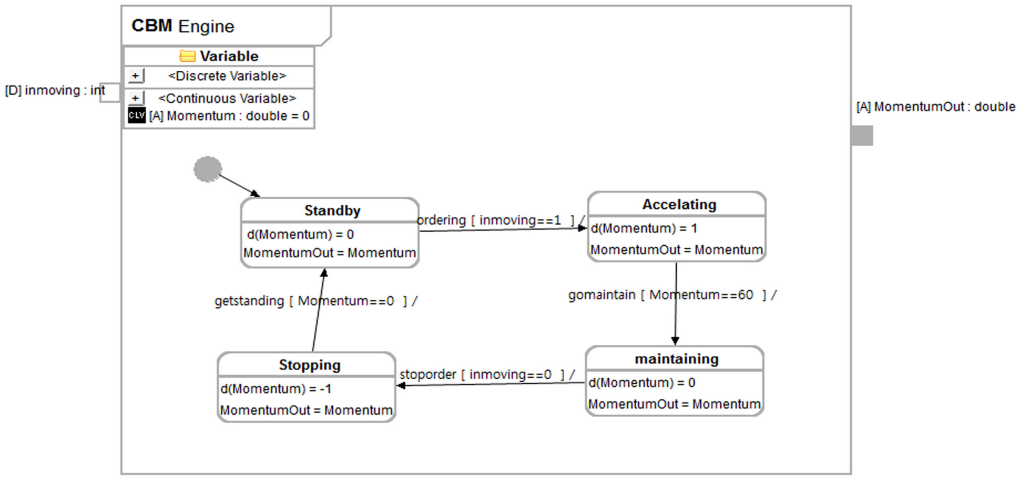

그림 9 ECML 행위 모델로 명세한 차량 모델의 일부(*Engine*)

Fig. 9 A part of vehicle model specified in ECML behavioral model (*Engine*)

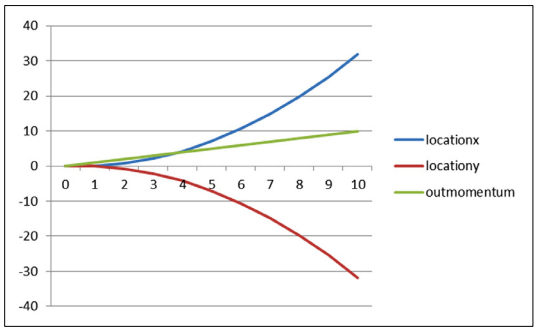

그림 10 자동차 모델의 위치 및 엔진 출력 변화

Fig. 10 A behavior about location and engine power of vehicle model

그림 10은 정의된 ECML 모델을 대상으로 목표 위치 (‘*destinationx, destinationy*’)를 (40, -40)으로 설정하였을 때 시뮬레이션 한 결과를 보여주고 있다. 차량 모델의 외부 입력과 출력에 해당하는 *‘locationx, locationy, outmomentum’*은 차량의 현재 위치와 엔진 출력을 나타내는 변수들로 ECML 환경모델에서 값을 정의해주고 받아들인다. 차량의 엔진 출력(‘*outmomentum*’)은 1씩 증가하며 이에 따라 차량의 현재 위치 변수들인 *‘locationx, locationy’*가 변화하는 것을 볼 수 있다. 환경 모델은 엔진출력과 차량의 진행방향에 따라 차량의 위치는 그림 5의 방식대로 sin, cos 함수를 이용하여 현재 위치를 계산하고 입력해준다.

5. 관련 연구

하이브리드 시스템은 컴퓨터 공학(computer science)와 제어 이론(control theory)이 연계되어 있으며 모델의 높은 안전성을 위해서는 검증 또한 두 관점의 검증 방식이 혼합되어야 한다[15,16]. 현재 하이브리드 오토마타(hybrid automata) 기반의 언어들은 컴퓨터 공학의 관점에서 모델체킹을, 시뮬레이션 언어들은 제어 이론의 관점에서 시뮬레이션 결과를 이용하여 검증을 수행하려고 하고 있다.

ECML은 시각적 모델링(visual model), 모듈화 및 계층성을 지원하는 시뮬레이션 언어이며 모델링 환경 및 시뮬레이션을 지원하는 도구가 개발되어 디자인된 모델에 대한 시뮬레이션을 이용한 검증이 수행되고 있다. ECML이 사용되는 하이브리드 시스템은 안전 필수 시스템에서 사용 될 수 있으므로 높은 안전성이 요구되며 기존의 시뮬레이션 언어들의 방식과 다르게 시뮬레이션 언어인 EMCL 모델체킹을 이용한 정형 검증을 추가로 수행해보려고 한다. 하이브리드 시스템을 사용하는 언어와 지원 도구는 시뮬레이션과 정형 검증의 목적에 따라 많은 문법적, 의미적 차이가 있음을 고려해야 한다. 특히 기본적으로 시뮬레이션 언어의 전이는 기본적으로 결정적 전이(deterministic transition)을, 하이브리드 오토마타 기반의 언어는 비결정적 전이(non -deterministic transition)을 사용하는 차이점과 하이브리드 시스템의 모델체킹을 이용한 검증은 상태 폭발 문제(state explosion problem)가 이산 시스템으로만 구성된 시스템을 검증하는 것보다 더 큰 문제가 되기 때문에 언어와 도구별로 변수의 종류나 크기, 표현 등을 매우 한정적으로 사용하고 있고 approximation [17,18] 등을 수행한다. 이러한 차이가 각각의 언어들에 반영되어 있다.

시뮬레이션과 정형 검증의 관점에 따라 언어 자체의

차이뿐만 아니라 모델을 만드는 사람의 모델링방식도 다를 수 있음을 또한 고려해야 한다. 예를 들어, 시뮬레이션을 위한 언어인 ECML의 경우 모델이 실행되는 환경, 즉 위치나 센서 입력 등을 환경 모델이라는 부분에서 전담하고 계산된 값만 일정하게 넣어주는 것을 가정하여 모델을 만드는 반면에 정형 검증을 목적으로 하는 언어는 무작위의 입력이 들어오고 들어올 수 있는 입력을 제약하는 방식으로 모델링을 하는 차이점이 있다.

ECML을 정형 검증의 용도로 사용하기 위해 우리의 기존의 연구인 [19]에서는 ECML의 기반 형식론인 DEV&DESS 모델을 linear hybrid automata로 변환 후 HyTech를 이용하여 검증해보았으며 [20]와 [21]에서는 단순한 ECML 모델을 변환하여 각각 HyTech와 SpaceEx 로 검증해 보았다. 이 후 ECML 모델을 기존의 정형 검증 도구를 이용하여 검증하기 위해 변환 과정을 자동화하는 도구를 개발하고 다양한 사례 연구를 수행할 계획이다.

6. 결 론

하이브리드 시스템은 연속 시스템과 이산 시스템이 혼합된 시스템이며 이를 명세하기 위한 다양한 언어 및 도구들이 제안되고 사용되어 왔다. ECML은 한국전자통신 연구원의 CPS 개발 환경에 맞게 제안된 하이브리드 시스템 명세언어이며 현재 모델링 및 시뮬레이션을 지원하는 도구가 개발되어 사용되고 있다. 논문에서는 ECML에 대한 정의와 특징을 설명하였으며, 사례연구를 통해 제안한 정의로 ECML 모델을 표현할 수 있음을 보였다. 하이브리드 시스템이 사용되는 환경은 안전성이 높기 요구되는 경우가 많기 때문에 이 후, 논문에서 제안한 정의를 기반으로, ECML로 명세 된 모델을 정형 검증의 용도로 사용하기 위한 연구를 추가적으로 진행할 것이다. 이를 위해 시뮬레이션과 정형 검증의 두 분야의 관점의 차이점들을 고려할 필요가 있다.

References

[1] A. Tewari, "Modern Control Design: with MATLAB and SIMULINK," Wiley Chichester, 2002.

[2] R. Alur, R. Grosu, Y. Hur, V. Kumar, I. Lee, "Modular Specification of Hybrid Systems in Charon," *Hybrid Systems: Computation and Control*, pp. 6-19, 2000.

[3] R. Alur, D. L. Dill, "A Theory of Timed Automata," *Theoretical Computer Science*, Vol. 126, No. 2, pp.183-235, 1994.

[4] K. G. Larsan, P. Pettersson, W. Yi, "Uppaal in a Nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, Vol. 1, No. 1, pp. 134-152, 1997.

[5] E. Asarin, T. Dang, O. Maler, "The d/dt tool for Verification of Hybrid Systems," *Computer Aided Verification*, pp. 365-370, 2002.

[6] R. Alur, C. Courcobetis, N. Halbwachs, T. A. Henzinger, P.-H, Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, "The Algorithmic Analysis of Hybrid Systems," *Theoretical Computer Science*, Vol. 138, No. 1, pp. 3-34, 1995.

[7] R. Alur, T. A. Henzinger, P.-H. Ho, "Automatic Symbolic Verification of Embedded Systems," *IEEE Transaction on Software Engineering*, Vol. 22, No. 3, pp. 181-201, 1996.

[8] N. Lynch, R. Segala, F. Vaandrager, "Hybrid I/O Automata," *Information and Computation*, Vol. 185, No. 1, pp. 105-157, 2003.

[9] T. A. Henzinger, P.-H. Ho, H. Wong-Toi, "HyTech: A Model Checker for Hybrid Systems," *Software Tools for Technology Transfer*, Vol. 1, pp. 110-122, 1997.

[10] G. Frehse, "PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech," *Hybrid Systems: Computation and Control, LNCS*, Vol.3414, pp. 258-273, 2005.

[11] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, "SpaceEx: Scalable Verification of Hybrid Systems," *Computer Aided Verification (CAV) 2011, LNCS*, Vol. 6806, pp. 379-395, 2011.

[12] H. Y. Lee, J. M. Kim, I. Chun, W.-T Kim, S-M Park, "Visual Modeling Language for Hybrid Systems Modeling," *Proc. of the 7th Conference on National Defense Technology*, pp. 884-890, 2001. (in Korean)

[13] B. P. Zeigler, H. Praehofer, T. G. Kim, "Theory of Modeling and Simulation," *Academic Press*, 2000.

[14] P. Derler, E. A. Lee, A. S. Vincentelli, "Modeling Cyber-Physical Systems," *Proc. of the IEEE*, Vol. 100, No. 1, pp. 13-38, 2012.

[15] H. Fang, J. Guo, H. Zhu, J. Shi, "Formal Verification and Simulation: Co-verification for Subway Control Systems," *Theoretical Aspects of Software Engineering (TASE), 2012 Sixth International Symposium on, IEEE,* pp. 145-152, 2012.

[16] A. Girard, G. J. Papas, "Verification using Simulation," *Hybrid Systems: Computation and Control*, Springer, pp. 272-286, 2006.

[17] E. Asarin, O. Bournez, T. Dang, O. Maler, "Approximate Reachability Analysis of Piecewise-Linear Dynamical Systems," *Hybrid Systems: Computation and Control*, pp. 20-31, 2000.

[18] T. A. Henzinger, H. Wong-Toi, "Linear Phase-Portrait Approximations for Nonlinear Hybrid Systems," *Hybrid Systems III, Springer Berlin Heidelberg*, pp. 377-388, 1996.

[19] H. Choi, S. Cha, J. Jo, J. Yoo, H. Y. Lee, W.-T. Kim, "Formal Verification of Basic DEV&DESS

Formalism using HyTech," *Information Journal*, Vol. 16, No. 1(B), pp. 821-826, 2013.
[20] J. Jo, J. Yoo, H. Choi, S. Cha, H. Y. Lee, W.-T. Kim, "Translation from ECML to Linear Hybrid Automata," *International Workshop on Technologies and Applications for Cyber Physical System (TACPS 2012/EMC-12), LNEE* 181, pp. 293-300, 2012.
[21] J. Jo, S. Yoon, J. Yoo, H. Y. Lee, W.-T. Kim, "Case Study: Verification of ECML Model Using SpaceEx," *Korea-Japan Joint Workshop on ICT*, pp. 1-4, 2012.

윤 상 현

2010년 건국대학교 컴퓨터공학부 졸업(학사). 2012년 건국대학교 대학원 컴퓨터·정보통신공학과 졸업(석사). 2012년~현재 건국대학교 컴퓨터·정보통신공학과 박사과정. 관심분야는 정형검증, 하이브리드 시스템

전 인 걸

1997년 성균관대학교 전자전기컴퓨터공학과 졸업(석사). 2010년 성균관대학교 전자전기컴퓨터공학과 졸업(박사). 1998년~현재 한국전자통신연구원 책임연구원. 2012년~현재 과학기술연합대학원대학교 겸임교수. 관심분야는 CPS, 스마트 팩토리, autonomic computing system, 임베디드 시스템, 소프트웨어 공학

김 원 태

1994년 한양대학교 전자공학과 졸업(학사). 1996년 한양대학교 전자공학과 졸업(석사). 2000년 한양대학교 전자공학과 졸업(박사). 2001년~2005년 (주)로스틱테크놀로지 CTO. 2010년~현재 한국전자통신연구원 임베디드SW연구부 CPS연구실 실장. 관심분야는 CPS, IoT, 임베디드 소프트웨어, 고신뢰 네트워크

조 재 연

2013년 건국대학교 대학원 컴퓨터·정보통신공학과 졸업(석사). 2013년 3월~2014년 4월 한양대학교 연구원. 2014년 5월~현재 한국전자통신연구원 연구원

유 준 범

2005년 KAIST 전자전산학과 전산학전공 졸업(박사). 2005년~2008년 삼성전자주식회사 통신연구소 책임연구원. 2008년~현재 건국대학교 컴퓨터공학부 부교수. 관심분야는 소프트웨어 공학, 안전성 분석, 정형기법