KIPS Trans. Comp. and Comm. Sys.
Vol.11, No.9 pp.289~304
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2022.11.9.289

GTS-Visual Logic: Visual Logic and Tool for Analysis and Verification of Secure Requirements in Smart IoT Systems

SungHyeon Lee† · MoonKun Lee††

ABSTRACT

It is necessary to apply process algebra and logic in order to analyze and verify safety requirements for Smart IoT Systems due to distributivity and mobility of the systems over some predefined geo-temporal space. However the analysis and verification cannot be fully intuitive over the space due to the fact that the existing process algebra and logic are very limited to express the distributivity and the mobility. In order to overcome the limitations, the paper presents a new logic, namely for GTS-VL (Geo-Temporal Space-Visual Logic), visualization of the analysis and verification over the space. GTS-VL is the first order logic that deals with relations among the different types of blocks over the space, which is the graph that visualizes the system behaviors specified with the existing dTP-Calculus. A tool, called SAVE, was developed over the ADOxx Meta-Modeling Platform in order to demonstrate the feasibility of the approach, and the advantages and practicality of the approach was shown with the comparative analysis of PBC (Producer-Buffer-Consumer) example between the graphical analysis and verification method over the textual method with SAVE tool.

Keywords : GTS-VL, dTP-Calculus, IoT, SAVE, ADOxx

GTS-VL: 스마트 IoT에서 안전 요구사항 분석과 검증을 위한 시각화 논리 언어 및 도구

이 성 현† · 이 문 근††

요 약

스마트 IoT의 특징인 분산성 및 이동성에 기반한 안전 요구사항을 분석 및 검증하기 위한 프로세스 대수 및 논리가 요구된다. 하지만 기존의 프로세스 대수 및 논리는 분산성 및 이동성에 대한 표현이 제한적이므로 스마트 IoT의 요구사항 분석 및 검증이 비직관적이다. 이러한 한계를 극복하기 위해, 본 논문에서는 GTS-VL(Geo-Temporal Space-Visual Logic)을 제시한다. GTS-VL은 GTS에서 표현된 블록 간의 관계를 다루는 1차술어논리이며, GTS는 프로세스 대수인 dTP-Calculus를 사용하여 명세한 시스템의 동작 과정을 2차원 시공간에서 표현한 그래프이다. 본 논문에서 사용한 SAVE 도구는 ADOxx Meta-modeling Platform을 통해 개발되었으며, SAVE를 사용하여 PBC(Producer-Buffer-Consumer) 예제의 안전 요구사항을 분석 및 검증하고 문자 및 시각화 기반 검증 방법을 비교 분석하여 장점 및 실용성을 보인다.

키워드 : GTS-VL, dTP-Calculus, IoT, SAVE, ADOxx

1. 서 론

4차 산업혁명의 주요 목표 중 하나는 스마트 IoT를 기반으로 한 자동화이며, 이를 위해 AI 및 빅데이터 등의 첨단 기술이 요구되는 만큼, 스마트 IoT의 안전성 및 보안성에 대한 요구사항이 강화되고 있다[1].

이러한 안전성과 보안성을 명세, 분석 및 검증하기 위해서는 정형기법이 요구되고 있으며, 스마트 IoT의 특성상, 프로세스의 분산성(Distrivutivity) 및 이동성(movement)이라는 특징 때문에 명세는 시각화에 기반을 둔 프로세스 대수(Process Algebra), 분석 및 검증은 시각화에 기반을 둔 논리(Logic)가 대세를 이루고 있다[2]. 그 이유는 명세 관점에서, 시각화에 기반을 둔 프로세스 대수가 문자만을 사용하여 명세하는 프로세스 대수보다 이동성 및 분산성을 표현하기 직관적이기 때문이다[2]. 또한 분석 및 검증 관점에서도 마찬가지로 시각화에 기반을 둔 논리 언어가 문자만을 사용하여 분석 및 검증을 수행하는 논리 언어보다 직관적이기 때문에, 스마트 IoT의 분산성과 이동성이라는 특징에 기초하여 시스템을 분석 및 검증할 수 있다.

일반적으로 스마트 IoT에 대한 요구사항은 크게 기능 요

† 준 회 원 : 전북대학교 컴퓨터공학부 박사
†† 비 회 원 : 전북대학교 컴퓨터공학부 교수
Manuscript Received : March 21, 2022
Accepted : May 24, 2022
* Corresponding Author : MoonKun Lee(moonkun@jbnu.ac.kr)

구사항과 안전 요구사항 두 가지로 분류된다. 대부분의 기능 요구사항은 프로세스 대수에 의해 명세되며, 안전 요구사항은 1차술어논리(*First-order logic*)에 기반한 논리 언어들을 사용하여 분석 및 검증된다[3].

스마트 IoT의 기능 요구사항에 대한 시각 명세 기능을 보여주는 몇몇 논리 언어들이 있지만[4, 5], 안전 요구사항에 대한 시각화 기능을 제공하는 논리 언어는 거의 없고, 스마트 IoT와 같이 이동성 및 분산성이 강조되는 시스템에 대한 안전 요구사항을 분석 및 시각화하기에는 여러 제약점이 있다[2].

본 논문은 위에서 언급한 논리 언어들의 단점을 보완하기 위해 GTS-VL (*Geo-Temporal Space-Visual Logic*)을 제시한다. GTS-VL은 Fig. 1과 같은 이중 접근법(Dual Approach)을 통해 스마트 IoT의 안전 요구사항을 분석 및 검증하기 위해 사용된다. Fig. 1의 이중 접근법은 다음과 같이 네 단계로 구성된다:

1) 시스템의 기능 요구사항을 프로세스 대수로 명세한다. 시각 명세 및 분석이 가능한 프로세스 대수인 dTP-Caluclus를 사용한다.
2) 명세한 기능 요구사항의 실행 가능한 모든 경우의 수를 분석하여 시뮬레이션을 수행한다. 시뮬레이션을 수행한 후, 모든 실행 가능한 경우의 수를 출력하는 실행모델을 생성한다.
3) 각 실행분기는 GTS (*Geo-Temporal Space*)를 통해 시각화되며, GTS-VL을 통해 시스템의 안전 요구사항을 도출한다.
4) 마지막으로 GTS-VL를 통해 시스템의 안전 요구사항 분석 및 검증하고, 검증 결과를 시각화하여 확인한다.

GTS는 모든 프로세스의 각 동작 및 프로세스 간의 동기 통신 및 상호작용, 프로세스의 동작의 종속성 및 포함 관계 등을 시각화하여 나타내는 논리이다. 시스템에서 각 프로세스들이 수행하는 동작 및 프로세스 간의 포함 관계는 GTS 상에서 블록으로 표현되며, GTS-VL는 GTS 상에서 표현된 각

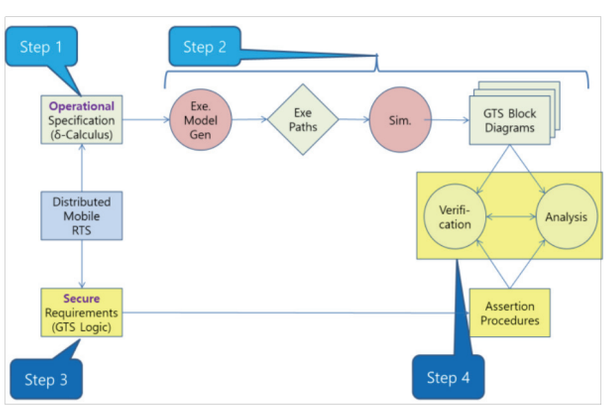


Fig. 1. A Dual Approach for Analysis and Verification of Requirements of Smart IoT

블록들의 포함 관계 및 시간 관계를 분석 가능한 1차술어논리이다. GTS-VL은 블록들의 시간 및 공간 관계를 분석하기 위한 다양한 술어(Predicates)를 제공한다. 시간 술어(Temporal predicate)는 블록들 간의 시간 관계, 공간 술어(Geo predicate)는 블록 간의 포함 관계, 마지막으로 상호작용 술어(Interaction predicate)는 프로세스 간의 동기 상호 작용에 대한 관계를 분석할 수 있다. 결과적으로 이러한 분석을 통해 시스템의 안전 요구사항을 검증할 수 있다. 또한 GTS-VL은 안전 요구사항의 분석 및 검증 결과를 시각화할 수 있다. 일반적으로 시각에 기반한 논리는 문자에 기반한 논리에 비해, 논리의 분석 과정에서 발생하는 분석의 복잡도를 감소시킬 수 있다[6-8].

스마트 IoT에 대한 명세, 분석 및 검증을 위한 시각화 논리 언어 GTS-VL의 유용성을 확인하기 위해, (Fig. 1)의 이중 접근법(*Dual Approach*)을 적용한 검증 도구인 SAVE를 사용할 수 있다. SAVE는 ADOxx 메타-모델링 도구[9]를 기반으로 저자가 개발한 검증 도구이다.

본 논문에서는 위의 이중 접근법을 적용한 도구인 SAVE를 사용하여 IoT 기반 시스템에 대한 안전 요구사항의 분석 및 검증을 수행한다. GTS-VL의 시각화 기반 검증 결과 및 일반 논리의 문자 기반 검증 결과에 대한 각 복잡도를 비교하고, 이동성 및 분산성이 두드러지는 IoT 기반 시스템에 대한 안전 요구사항을 분석 및 검증 할 때, GTS-VL에 기반한 검증 방법의 우수성을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 선행 및 관련 연구를 통해 기존의 시간 논리 및 공간 논리의 특징과 장단점을 살펴본다. 3장에서는 GTS-VL에 대한 정의를 살펴본다. 4장에서는 GTS-VL을 사용하여 PBC 예제의 안전 요구사항을 명세 및 검증한다. 5장에서는 비교 분석을 통해 문자에 기반한 요구사항 분석과 시각화에 기반한 요구사항 분석의 복잡도를 비교한다. 6장에서는 SAVE 도구에 대해 소개한다. 7장에서는 결론 및 향후 연구 방향에 대해 논의한다.

2. 선행 및 관련 연구

2.1 선행연구

스마트 IoT를 명세하기 위한 프로세스 대수로써 dTP-Calculus가 있다. dTP-Calculus의 속성은 이동성, 동기성, 시간, 우선순위 총 5가지로 구성되며, 시스템을 구성하는 프로세스는 위의 속성에 기반하여 명세가 된다. 이동성은 *in*, *out*, *get*, *put*이라는 네 가지 이동 유형에 따라 표현되며, 이동을 위해서는 프로세스 간의 연결된 채널을 통해 동기 통신이 먼저 수행되어야 한다. dTP-Calculus의 문법은 Fig. 2와 같다.

dTP-Calculus에 의해 시스템을 명세 후에, 시뮬레이션을 통해 시스템의 모든 실행 분기를 분석 및 도출 할 수 있다. 도출된 결과는 SAVE 도구에서 실행모델로써 생성된다. 실행모델은 시스템이 수행 가능한 모든 실행 과정을 트리 형태로

$P ::= A$	Action	(1)	$A ::= \phi$	Empty	(16)
$\mid A^{per,n}_{[r,to,e,d]}$	Timed Action	(2)	$\mid ch(\overline{msg})$	Send	(17)
$\mid P^{per,n}_{[r,to,e,d]}$	Timed Process	(3)	$\mid ch(msg)$	Receive	(18)
$\mid P_{(pri\_n)}$	Priority	(4)	$\mid M$	Movement	(19)
$\mid P[Q]$	Nesting	(5)	$\mid C$	Control	(20)
$\mid P\langle ch\rangle$	Channel	(6)	$M ::= m^{pri}(k)\ P$	Movement Request	(21)
$\mid P + Q$	Choice	(7)	$\mid P\ m(k)$	Movement Permission	(22)
$\mid P\{pc\} +_F Q\{pc\}$	Probabilistic Choice	(8)	$m ::= in$	In Movement	(23)
$\mid P \parallel Q$	Parallel	(9)	$\mid out$	Out Movement	(24)
$\mid P\backslash E$	Exception	(10)	$\mid get$	Get Movement	(25)
$\mid A \cdot P$	Sequence	(11)	$\mid put$	Put Movement	(26)
$F ::= D$	Discrete Distribution	(12)	$C ::= new\ P$	Create Process	(27)
$\mid N(\mu,\sigma)$	Normal Distribution	(13)	$\mid kill\ P$	Kill Process	(28)
$\mid Ex(\lambda)$	Exponential Distribution	(14)	$\mid exit$	Exit Process	(29)
$\mid U(l,u)$	Uniform Distribution	(15)			

Fig. 2. Syntax of dTP-Calculus[2]

보여주며, 시스템의 기능 요구사항을 분석할 때 사용된다.

dTP-Calculus에 기반한 SAVE 도구에서는 각 실행 분기의 세부 분석을 위해 GTS를 생성할 수 있다. GTS는 2차원 공간 내에서 시스템의 프로세스가 수행한 행동 및 상호작용의 결과를 시각화하여 보여주는 그래프이다. GTS에서 x축은 시간 공간을 나타내며, y축은 지리 공간을 나타낸다. GTS는 시스템 및 시스템의 구성 요소 및 행동들을 블록의 형태로 표현한다. 블록의 종류는 시스템 블록, 프로세스 블록, 행동 블록, 상호작용 블록들이 있다. GTS에서 각 블록들의 위치와 연결선을 통해 시스템 내에서 프로세스들의 이동 및 상호작용을 시각화 가능하다.

2.2 관련연구

시간 논리는 LTL (*Linear Temporal Logic*)[10], CTL (*Computational Tree Logic*)[11], RTL (*Real-time Logic*)[12] 등이 있다. CTL 및 LTL의 경우 시간 연산자를 사용한 시제 논리를 구성한다. RTL의 경우 이벤트와 상태 발생 함수를 통해 시제 논리를 전개한다. 위와 같은 시간 논리들은 시간을 분석할 때는 유용하나, 스마트 IoT와 같이 시간과 공간 및 공간에 대한 이동성을 동시에 표현하기에는 직관적이지 못하다.

대표 공간 논리로는 RCC(*Region and Connection Calculus*)[13], CRD(*Cardinal Direction Relations*)[14] 등이 있다. RCC는 공간 간의 관계를 정의하여 공간 논리를 전개하며, CRD는 좌표에 기반한 공간 논리를 구성한다. 이러한 언어들은 시간에 대한 명세를 할 수 없으며, 시간에 따른 이동성의 표현에 제한되기 때문에 스마트 IoT를 구성하는 프로세스의 이동성을 분석할 수 없다는 단점을 지니고 있다.

또 다른 논리로는 1차술어논리에서 사용되는 각 기호 및 의미를 시각화하기 위한 VL (Visual Logic)이 있다[15]. VL은 1차술어논리에서 사용되는 술어, 양화사, 변수 및 상수를 도형과 연결선 등으로 시각화하여 표현한다. VL에서 다루는 1차술어논리의 시각화는 1차술어논리에서 사용되는 기호 및 술어 간의 관계에만 한정되어 있다.

GTS-VL은 GTS안에 존재하는 두 개의 블록 혹은 단일 블록의 시간 공간 및 지리 공간의 관계에 대해 논리를 기반으로 명세할 수 있는 1차술어논리이며, 시스템의 안전 요구사항을 분석 및 검증 가능하다. 여기서 GTS는 프로세스의 동기화 및 이동에 기반한 시스템의 진행 과정을 시간에 따라 블록의 형태로 보여주며, x축은 시간 공간, y축은 지리 공간을 나타낸다. GTS의 각 공간을 구성하는 블록은 시스템 블록, 프로세스 블록, 행동 블록, 상호작용 블록으로 나뉜다. GTS-VL을 통해 요구사항을 검증한 결과는 GTS 상에서 연결선 및 연결선의 색상을 통해 시각화되기 때문에, 기존의 시간 및 공간 논리 언어에 비해 시스템의 요구사항 분석 및 검증을 보다 간편 및 명확하게 할 수 있다는 장점을 지니고 있다.

3. GTS-VL

GTS-VL (*Geo-Temporal Space-Visual Logic*)은 GTS에서 표현된 블록들 간의 관계를 명세하는 1차술어논리이다. GTS-VL에서 시스템은 $S=<P,I,C>$로 정의되며, 여기서 P,I,C는 각각 프로세스, 포함 관계 및 채널의 집합이다. 각 P는 Fig. 2에 명시된 dTP-Calculus 문법에 기반하여 시간 제한 동작(*timed action*)에 의해 정의된다[2].

시스템에서 통신 및 이동은 다음과 같은 각 프로세스 간의 동기 상호 작용에 의해 발생한다:

1) Send/Receive: 프로세스 간 통신을 의미한다. 채널 r을 통해 프로세스 간에 메시지를 교환한다.
2) Movement request: 프로세스의 이동을 요청한다. 여기서 p와 k는 각각 우선순위와 키(key)를 나타낸다.
3) Movement permission: 프로세스의 이동을 허가한다. timed action의 $[r,to,e,d]$는 각각 준비 시간, 타임아웃, 실행시간, 마감 시간을 나타낸다. p 및 n은 각각 프로세스의 반복 주기와 반복 횟수를 나타낸다.

시스템이 특정 시간 및 공간 내에서 실행이 될 때, 시스템 내에 있는 각 프로세스들의 동작 및 상호작용에 대한 모든 실

행 가능한 경우를 시뮬레이션 및 생성한다.

이러한 결과는 GTS를 통해 시각화되어 표현될 수 있다. GTS는 지리 공간과 시간 공간 두 가지 차원으로 구성되며, 세로 축은 지리 공간, 가로 축은 시간 공간을 나타낸다. GTS에는 세 가지 유형의 블록이 존재한다. 세 가지 블록은 시스템 블록(S), 프로세스 블록(P), 행동 블록(A)이 있다. GTS의 정의에 따르면, 시스템 블록에는 프로세스 블록이 포함되고, 프로세스 블록에는 행동 블록이 포함된다. 또한 GTS에서 상호작용은 서로 다른 두 프로세스 간의 상호작용 블록(I)에 의해 표시된다. GTS에서 표현된 블록들은 GTS-VL의 술어에 기반하여 공간 및 시간 관계 등을 검증할 수 있다.

3.1 GTS-VL의 문법

3.1.1 GTS-VL의 알파벳

1) 논리 부호:
   - GTS-VL은 일반 술어논리에서 사용하는 논리부호를 사용한다. ($\wedge, \vee, \rightarrow, \leftrightarrow, =$ 등)

2) 비논리 부호:
   ① 술어 부호: $P, Q, R(P_j^i)$
   ② 함수 부호: $f, g, h\,(f_j^i)$
   ③ 상수: a, b, c
   - 정수: 0, 1, 2, ...
   - 문자: “a”, “aa”, ..., “abc”, ...

3.1.2 GTS-VL의 구문

1) 항 (Terms):
   ① 변수: x, y, z와 같은 변수들은 항이다.
   ② 함수: $t_1, t_2, ..., t_n$이 모두 항이고, f가 n개의 인자를 갖는 함수 기호일 때, $f(t_1, t_2, ..., t_n)$은 항이다.

2) 정식 (Formula):
   ① $t_1, t_2, ..., t_n$이 모두 항이고, P가 n개의 인자를 갖는 술어 기호일 때, $P(t_1, t_2, ..., t_n)$은 정식이다.
   ② P와 Q가 정식이면, 논리 기호를 사용하여 구성되는 논리식 $\neg P, P \vee Q, P \wedge Q, P \rightarrow Q, P \leftrightarrow Q$도 정식이다.
   ③ $P(x)$가 정식이고, x가 변수일 때, $(\exists x)P(x), (\forall x)P(x)$는 정식이다.

3.2 GTS-VL의 함수

GTS-VL의 함수는 다음과 같다:

1) $begin(x)$: x 블록의 시작 시간을 반환한다.
2) $end(x)$: x 블록의 종료 시간을 반환한다.
3) $name(x)$: x 블록의 이름을 반환한다.
4) $time\_relation(x,y)$: x와 y 블록 간의 시간 관계 값을 반환한다.
   : *next*, *prev*, *simul_st*, *simul_ed*, *before*, *after*
5) $geo\_relation(x,y)$: x와 y 블록 간의 공간 관계 값을 반환한다.
   : *parent*, *child*, *parallel*
6) $info(x)$: x의 상호작용에 필요한 정보를 반환한다
7) $next\_process(x)$, $prev\_process(x)$: 연속된 프로세스 블록의 다음 블록(x_{i+1})또는 이전 블록을 반환한다.
8) $next\_action(x)$, $prev\_action(x)$: 연속된 행동 블록의 다음 블록(x_{i+1})또는 이전 블록을 반환한다.
9) $interaction(x)$: 상호작용 블록을 반환한다. 여기서 x는 행동 혹은 상호작용 블록이다. 만약 상호작용블록이 아니라면 액션 블록을 반환한다.
10) $actor(x)$: 상호작용 블록의 행동 블록(*actor*)을 반환한다.
11) $actee(x)$: 상호작용 블록의 행동 블록(*actee*)을 반환한다.
12) $process(x)$: 어떤 행동 블록에 대한 프로세스 블록을 반환한다.
13) $children(x)$: 프로세스 x의 자식 프로세스의 개수를 반환한다.

3.3 GTS-VL의 술어

GTS-VL의 술어는 다음과 같이 네 가지로 구분되며, 블록에 대한 기본 술어를 기반으로 시간, 공간 및 상호작용 술어의 조합에 의해 안전 요구사항이 명세된다.

- 블록 술어 (*Block predicate*)
  : 블록에 대한 기본 공리를 정의하는 술어이다.
- 시간 술어 (*Temporal predicate*)
  : 시간 술어는 블록 단일 혹은 두 블록의 시간 관계를 명세하기 위해 사용된다.
- 공간 술어 (*Geo predicate*)
  : 공간 술어는 두 블록의 공간 관계를 명세하기 위해 사용된다.
- 상호작용 술어 (*Interaction predicate*)
  : 상호작용 술어는 시스템에서 상호작용의 발생 유무를 판단하기 위해 사용된다.

3.3.1 블록 술어

GTS-VL에서 다루는 블록에 대한 기본 공리는 다음과 같다:

1) $\forall x B(x)$
   : 모든 x는 블록이다.
2) $\forall x PB(x) \rightarrow B(x)$
   : 모든 프로세스 블록은 블록이다.
3) $\forall x AB(x) \rightarrow B(x)$
   : 모든 액션 블록은 블록이다
4) $\forall x PB(x) \leftrightarrow \neg AB(x)$
   : 프로세스 블록이면서 동시에 행동 블록일 수 없다.
5) $\forall x AB(x) \leftrightarrow \neg PB(x)$
   : 행동 블록이면서 동시에 프로세스 블록일 수 없다.

6) $\forall xyIB(x) \leftrightarrow AB(actor(x)) \wedge AB(actee(x)) \wedge (info(actor(x)) = info(actee(x)))$

: 상호작용 블록 x는 상호작용 블록에 대한 *actor*와 *actee*에 대한 액션 블록 있어야하며, *actor*와 *actee* 동기화에 대한 정보가 같아야 한다. 여기서 *actor*는 상호작용을 수행하는 수행자이며, *actee*는 상호작용을 허가하는 수행허가자이다.

3.3.2 시간 술어

시간 술어는 시스템의 시간 구간을 명세하기 위해 사용되며, 시간 술어에 대한 정의는 다음과 같다:

1) $\forall xy \exists r\, TIME(x,y,r) \leftrightarrow B(x) \wedge B(y) \wedge (r = time\_relation(x,y))$

: 모든 블록은 시간 술어로 명세할 수 있다. 여기서 r은 문자열 상수이며 블록 간의 시간 관계에 대한 정보를 나타낸다. r은 종류는 *next, prev, simul_st, inclusion, simul_ed, before, after* 및 *equal*이며, 각 r의 종류의 의미를 시각화하면 (Fig. 3)과 같다.

또한 3.3.2의 1)의 $TIME(x,y,r)$로부터 다음과 같은 규칙을 정의할 수 있다.

2) $TIME(x,y,'next') \leftrightarrow\rightarrow \neg TIME(x,y,'prev')$

: *next*는 *prev*의 역이다.

3) $TIME(x,y,'before') \leftrightarrow\rightarrow \neg TIME(x,y,'after')$

: *before*는 *after*의 역이다.

4) $TIME(x,y,'equal') \leftrightarrow TIME(x,y,'simul\_st') \wedge TIME(x,y,'simul\_ed')$

: *equal*은 *simul_st* 및 *simul_ed*가 동시에 성립되어야 한다.

두 블록의 관계 외에 시간 상수를 사용하여 술어를 명세할 수 있다:

5) $\exists x BEFORE(x,t) \leftrightarrow B(x) \wedge (end(x) \leqq t)$

: 어떤 블록 x가 t시간 이전에 존재하면 참이다.

6) $\exists x AFTER(x,t) \leftrightarrow B(x) \wedge (t \leqq begin(x))$

: 어떤 블록 x가 t시간 이후에 존재하면 참이다.

7) $\exists x\, TICK(x,t) \leftrightarrow B(x) \wedge AFTER(x,t) \wedge BEFORE(x,t)$

: 어떤 블록 x가 명시한 시간에 있으면 참이다.

위의 5), 6) 및 7)을 통해 다음 술어를 정의할 수 있다:

8) $\exists x GLOBAL(x) \leftrightarrow B(x) \wedge AFTER(x,0) \wedge BEFORE(x,'end')$

: 어떤 블록 x가 시스템이 수행되는 시간 내에 존재하면 참이다. 여기서 *end*는 시스템의 수행 종료시간을 의미한다.

Type	Name	Description	Visualization of Predicate
TIME	before(P, Q)	P exists before the beginning of Q	
	prev(P,Q)	P exists before the beginning of Q. But P and Q are adjacent.	
	simul_st(P,Q)	P and Q are executed simultaneously.	
	simul_ed(P,Q)	P and Q are terminated simultaneously.	
	next(P,Q)	P exists after the end of Q. But P and Q are adjacent.	
	after(P,Q)	P exists after the end of Q	
	inclusion(P, Q)	Q exists during the execution of P	
	equal(P,Q)	P and Q exist at the same time	
	before(P,t)	P exists before time t	
	after(P,t)	P exists after time t	
GEO	parallel(P,Q)	P and Q are parallel	
	parent(P,Q)	P is a parent of Q	
	child(P,Q)	P is a child of Q	
Interaction	interaction(P,Q)	P and Q are interacted	

Fig. 3. Semantic and Visualization of Temporal and Geo Predicate

9) $STR\_TIME(x,t_1,t_n) \leftrightarrow B(x) \wedge TICK(t_1) \wedge TICK(t_{1+1}) \wedge \ldots \wedge TICK(t_n)$ $where\ t = \{t_1, t_{1+1}, \ldots, t_n\}$

: 어떤 블록 x가 시스템이 수행되는 시간 내에 존재하면 참이다. 다만 명시한 시간 구간 내에 모두 존재하여야 참이 된다.

3.3.3 공간 술어

공간 술어는 프로세스 간의 포함 관계를 명세하기 위해 사용된다. 공간 술어에 대한 정의는 다음과 같다:

1) $\forall xy \exists r\, GEO(x,y,r) \leftrightarrow B(x) \wedge B(y) \wedge (r = geo\_relation(x,y))$

: 모든 블록은 공간 술어로 명세할 수 있다. 여기서 r은 문자열 상수이며 블록 간의 공간 관계에 대한 정보를 나타낸다. r은 종류는 *parent, child* 및 *parallel*이며, 각 r의 종류의 의미를 시각화하면 Fig. 3과 같다.

3.3.3의 1)의 $GEO(x,y,r)$로부터 다음과 같은 규칙을 정의할 수 있다:

2) $GEO(x,y,'parallel') \leftrightarrow GEO(y,x,'parallel')$
: x와 y의 parallel 관계는 y와 x의 *parallel* 관계와 같다.
3) $GEO(x,y,'parent') \leftrightarrow \neg GEO(x,y,'child')$
: *child*의 역은 *parent*이다.

두 블록의 관계 외에 상수를 사용하여 술어를 명세할 수 있다:

4) $CAPA(x,c) \leftrightarrow children(x_1) \leqq \wedge childred(x_2) \leqq c \wedge \ldots \wedge childred(x_i) \leqq c where\ x = \{x_1, x_2, x_i\}$
: 어떤 프로세스 x의 자식의 개수는 c의 값 이하여야 참이다. 여기서 c는 0을 포함한 자연수이며, x는 하나의 프로세스이지만, 이동 상호작용에 의해서 GTS에서 분리되어 표현된 모든 프로세스 블록의 집합을 뜻한다.

3.3.4 상호작용 술어

시간 및 블록 술어를 사용하여 상호작용 블록에 대한 술어를 명세할 수 있다. 상호작용 술어는 술어를 통해 명세한 상호작용이 시스템에서 수행되었는지 판별한다.

1) $\exists x r INTERACTION(x,r) \leftrightarrow (IB(x) \vee PB(x) \vee AB(x)) \wedge GLOBAL(x) \wedge (info(x) = r)$
: 어떠한 상호작용이 시스템이 동작하는 시간 내에 존재해야하며, 상호작용 블록 동기화 정보 x는 r과 같아야 한다. 여기서 r은 통신, 이동, 제어에 해당한다.

GTS-VL에서는 이동은 *active-in(in)*, *active-out(out)*, *passive-in(get)* 및 *passive(put)* 총 네 가지가 있다. 각 이동에 대한 술어에 대한 공간 의미는 다음과 같다.

2) $\exists x r INTERACTION(x,'in') \rightarrow GEO(process(actor(x)), process(actee(x)), 'child')$
: *in* 상호작용 후에는 *actor*가 *actee*의 *child* 이어야 참이다.
3) $\exists x r INTERACTION(x,'out') \rightarrow GEO(process(actor(x)), process(actee(x)), 'parallel')$
: *out* 상호작용 후에는 *actor*와 *actee*가 *parallel* 관계여야 한다.
4) $\exists x r INTERACTION(x,'get') \rightarrow GEO(process(actor(x)), process(actee(x)), 'parent')$
: *get* 액션을 수행한 후에는 *actor*는 *actee*의 *parent*여야 한다.
5) $\exists x r INTERACTION(x,'put') \rightarrow GEO(process(actor(x)), process(actee(x)), 'parallel')$
: *put* 액션을 수행한 후에는 *actor*와 *actee*는 *parallel* 관계여야 한다.

또한 GTS-VL에서는 이동 상호작용을 수행하기 전에 먼저 이동 상호작용을 수행하려는 두 프로세스 간에 동기화 통신이 먼저 이루어져야 한다. 그러므로 다음과 같은 정의가 성립된다.

6) $INTERACTION(interaction(x), move) \rightarrow INTERACTION(interactino(prev\_action(x)), comm)$
: 이동 상호작용이 일어나면, 이동 상호작용이 일어나기 전에 통신에 의한 동기화가 먼저 수행되어야 한다.

위에서 언급한 시간 및 공간 술어에 대한 시각적인 의미는 Fig. 3과 같다.

3.4 GTS-VL 술어의 시각화

GTS-VL을 통해 명세한 요구사항, 즉 시공간 및 액션에 대한 술어는 시각화할 수 있다. 논리 언어의 시각화는 문자 기반의 요구사항 명세에 비해 가시성이 높기 때문에, 문자만 사용하여 시스템의 명세 결과를 분석하는 것보다 간결하게 시스템의 명세 결과를 분석할 수 있다.

GTS-VL은 논리 부호, 시간 술어, 공간 술어 및 상호작용 술어에 대한 총 네 가지 시각화 방법을 제공한다.

3.4.1 논리 기호의 시각화

이항 논리 기호에 대해, Fig. 4와 같이 연결선과 연결선의 중앙에 논리 기호를 표기하여 논리 기호를 시각화한다. 논리 기호의 연결선은 각 항에서 표현된 시각화된 술어와 연결되며, 요구사항의 만족 여부는 연결선의 색으로 표현된다. 파란색 연결선은 요구사항을 만족한다는 의미이며, 빨간색 연결은 요구사항을 만족하지 못한다는 의미를 가진다.

3.4.2 시간 술어의 시각화

시간 술어의 시각화는 Fig. 3의 술어 부호의 시각화 항목과 같이 각각의 시간 술어에 대해, 두 블록 사이의 연결선을

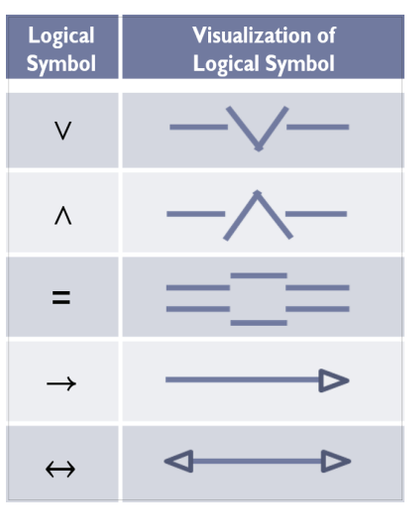
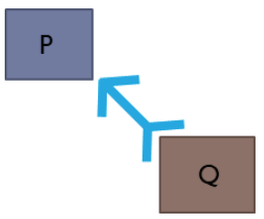
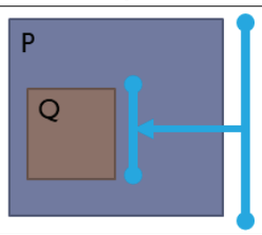
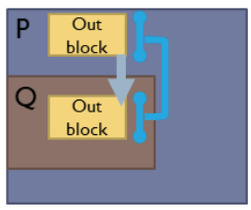


Fig. 4. Visualization of Logic Operators

Table 1. Example of Visualization for Temporal, Geo and Interaction Predicate

Type of Predicate	Time	Geo	Interaction
Example of Predicate	$TIME(P, Q, 'Before')$	$GEO(P, Q, 'Parent')$	$INTERACTION(P, 'out')$
Visualization of Predicate			

통해 표현된다. 만약 두 블록간의 시간 술어가 참이라면 파란색 연결선으로 표시되며, 거짓이라면 빨간색 연결선을 통해 표현된다. Table 1은 두 블록에 대해, *TIME* 술어 중 *Before*의 시각화 예시를 보여준다.

3.4.3 공간 술어의 시각화

공간 술어의 시각화는 Fig. 3의 술어 부호의 시각화 항목과 같이 두 프로세스가 포함 또는 평행 관계인지 판별할 수 있도록 막대와 연결선을 통해 시각화한다. 두 프로세스가 병렬 관계를 시각화할 때는 두 프로세스가 병렬임을 시각화하기 위해 아치형 연결선을 사용하여 표현하며, 두 프로세스가 포함 관계일 때는 화살표를 통해 프로세스가 포함 관계임을 표기한다. Table 1은 두 블록에 대해, *GEO* 술어 중 *Parent*의 시각화 예시를 보여준다.

3.4.4 상호작용 술어의 시각화

상호작용 술어의 시각화는 Fig. 3의 술어 부호의 시각화 항목과 같이 상호작용이 수행된 시점에서, 상호작용과 일치하는 행동 블록 및 막대를 통해 시각화한다. 만약 명세한 블록이 상호작용이 아닌 단일 행동의 경우, 단일 행동을 수행한 시점에서 명세한 행동과 일치하는 행동 블록에 막대로 표현한다. Table 1은 상호작용 블록에 대한 *INTERACTION* 술어의 시각화 예시를 보여준다.

4. 예 제

본 장에서는 스마트 IoT를 GTS-VL에 적용할 수 있음을 입증하기 위해 PBC (*Producer-Buffer-Consumer*) 예제를 기반으로 시스템을 명세, 분석 및 검증한다. PBC 예제의 명세, 분석 및 검증을 위한 도구로서 SAVE를 사용한다. PBC 예제의 동작 명세를 위해 dTP-Calculus를 사용한다. dTP-Calculus는 프로세스 대수 기반의 명세 언어이다.

4.1 요구사항

PBC 예제는 두 가지 종류의 요구사항이 있다.

1) 기능 요구사항:
- 생산자는 *R1* 및 *R2* 두 자원을 생산한다.
- 생산자는 버퍼에 *R1* 및 *R2*를 순서대로 저장한다.
- 생산자는 버퍼에게 *R1* 및 *R2*, 혹은 *R2* 및 *R1* 자원을 순서대로 저장했음을 알린다.
- 소비자는 버퍼로부터 *R1* 및 *R2*를 소비한다.
- 소비자는 버퍼가 전달하는 자원의 순서대로 *R1* 및 *R2*를 소비한다.

2) 안전 요구사항:
- 보안 정보 때문에 반드시 *R1*을 소비 혹은 전달한 후에 *R2*를 소비 혹은 전달해야 한다.
- 버퍼의 허용 가능한 최대 자원치는 2개이다.
- 생산자는 소비자에게 10 단위 시간 이내에 *R1* 및 *R2*를 전달해야한다.

4.2 명세(Specification)

Fig. 5는 PBC 예제를 dTP-Caculus의 문법에 따라 명세한 것이다. 예제에 대한 설명은 다음과 같다:

1) *PBC* 시스템은 세 개의 프로세스 *P*, *B*, *C*가 존재한다.

$$PBC = P[R1 \parallel R2] \parallel B \parallel C$$
$$P = \left(PB\left(\overline{Send\ R1}\right).put\ R1.put\ R2 + PB\left(\overline{Send\ R2}\right).put\ R2.put\ R1\right).exit$$
$$B = (PB(Send\ R1).get\ R1.get\ R2 + PB(Send\ R2).get\ R2.get\ R1).put\ R1.put\ R2.exit$$
$$C = get\ R1.get\ R2.exit$$
$$R1 = P\ put.B\ get.B\ put.C\ get.exit$$
$$R2 = P\ put.B\ get.B\ put.C\ get.exit$$

Fig. 5. Specification for PBC Example

2) *P*는 자식 프로세스인 *R1*과 *R2*를 가지고 있고, *P*와 *B* 사이에 *PB* 통신 채널, *B*와 *C* 사이에 *BC* 통신 채널을 가지고 있다.
3) *PBC* 시스템은 *P*에서 *B*로 자원을 보낼 때, 비결정 선택 분기(+)에 따라서 자원을 보낸다. 즉 *R1* 자원을 보낸 다음에 *R2* 자원을 보내거나, *R2* 자원을 보낸 후에 *R1* 자원을 보내게 된다.
4) 선택 분기에서 선택이 결정되면 *P*와 *B* 두 프로세스 간에 동기 이동 행동이 발생하여 선택된 행동에 따라서 순서대로 자원을 프로세스에게 전달하게 된다.
5) 먼저 *put* 행동을 통해 *P*가 자원을 순서대로 밖으로 내보낸다. 그 후에 *B*가 *get* 액션을 통해 자원을 순서대로 받게 된다.
6) 그 후에 *B*가 *C*로 자원을 보낼 때, 비결정 선택 분기에 따라서 다시 자원을 보내게 된다. *R1*을 먼저 보낸 후에 *R2*를 보내거나, *R2*를 보낸 후에 *R1*을 보내게 된다.

4.3 분석(Analysis)

Fig. 5의 dTP-Calculus 명세로부터 실행 가능한 모든 경로를 탐색하여, 모든 실행의 가짓수를 출력하면 Fig. 6과 같이 트리 형태로 시각화할 수 있다. Fig. 6은 dTP-Calculus 및 GTS-VL에 기반한 SAVE 도구로부터 생성된 것이다.

각 원형 아이콘은 위쪽에서부터 시간 순으로 나열되어 있으며, 각각은 시스템 상태 정보를 저장하고 있다. 시스템에서 각각의 원형 아이콘 옆에 있는 문자는 프로세스의 포함관계를 나타낸다. 예를 들어 case 01의 가장 첫 번째 원형 아이콘 옆에 있는 문자 $B[P[R1|R2]]|C$는 P, B 및 C가 서로 같은 공간 내에 있고, $R1$ 및 $R2$는 P 안에 포함되어 있다는 의미이다.

원형 아이콘과 연결된 연결선 옆에 있는 문자의 의미는 시스템의 상태가 다음 상태로 전이될 때 발생한 상호작용을 의미한다. 상호작용의 종류로는 프로세스의 통신에 의한 동기화 및 프로세스의 이동이 있다. 만약 상호작용으로 프로세스의 이동이 일어날 경우, 다음 상태로 전이될 때 프로세스의 포함 관계가 변경된다.

시스템의 상태는 Fig. 6과 같이 분기가 일어날 수 있다. Fig. 6을 살펴보면 PBC 예제는 총 4가지 실행 경로가 있음을 확인할 수 있다. 만약 deadlock이 발생하였을 경우, 원형 아이콘의 색깔이 빨간색으로 변경된다. Fig. 6의 경우 deadlock없이 네 가지 실행 경로가 모두 정상 실행되었음을 확인할 수 있다.

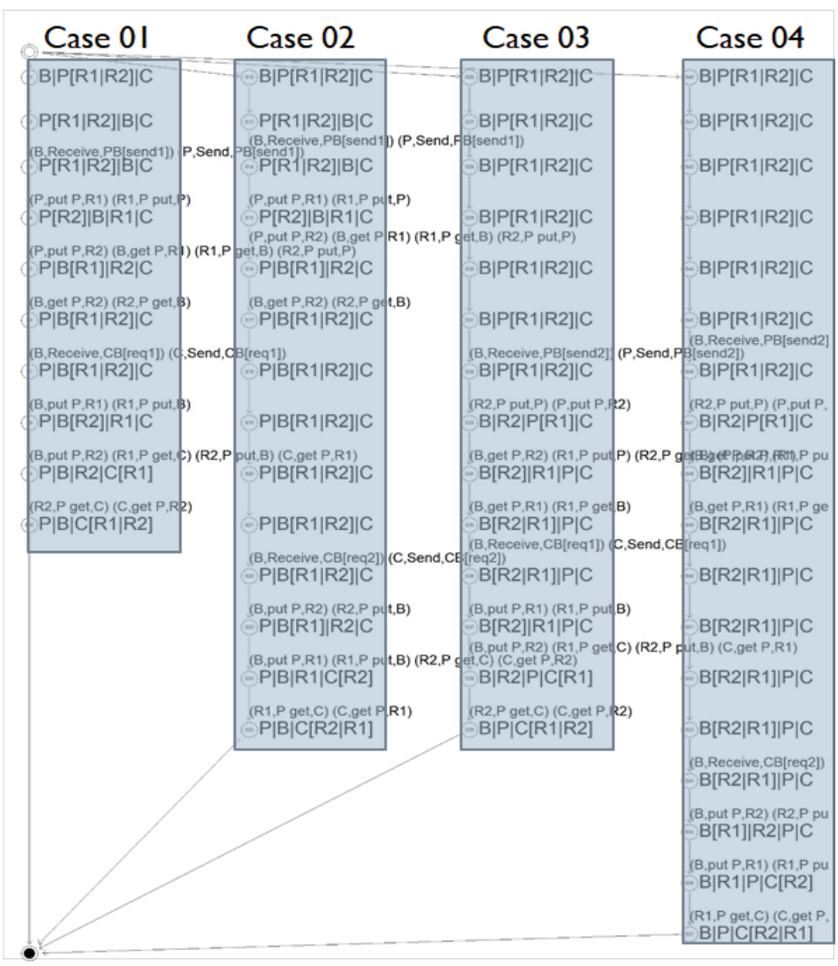


Fig. 6. Execution Model of PBC Example

Table 2. List of the Interaction for The Fourth Execution Path

Inter action	Semantic
$\delta 1$	*P* out *R2* in 7 time units
$\delta 2$	*B* get *R2* in 8 time units
$\delta 3$	*P* out *R1* in 8 time units
$\delta 4$	*B* get *R1* in 9 time units
$\delta 5$	*B* out *R2* in 15 time units
$\delta 6$	*B* out *R1* in 16 time units
$\delta 7$	*C* get *R2* in 16 time units
$\delta 8$	*P* out *R2* in 17 time units
$\tau 1$	*P* communicate with *B* in 6 time units
$\tau 2$	*B* communicate with *C* in 14 time units

Fig. 6의 가장 왼쪽 분기부터, 시스템이 수행한 행동을 요약하면 다음과 같다:

- *P*가 *R1* 및 *R2* 순서대로 *B*에게 보냄. *B*는 *R1* 및 *R2* 순서대로 *C*에게 보냄
- *P*가 *R1* 및 *R2* 순서대로 *B*에게 보냄. *B*는 *R2* 및 *R1* 순서대로 *C*에게 보냄
- *P*가 *R2* 및 *R1* 순서대로 *B*에게 보냄. *B*는 *R1* 및 *R2* 순서대로 *C*에게 보냄
- *P*가 *R2* 및 *R1* 순서대로 *B*에게 보냄. *B*는 *R2* 및 *R1* 순서대로 *C*에게 보냄

각 실행 경로의 세부 분석을 위해 실행 경로 중 하나를 선택하여 GTS를 생성할 수 있다. GTS란 프로세스의 동기화 및 이동을 2차원 공간에서 블록으로 나타낸 그래프이다. GTS에 표현된 각 블록들의 의미를 파악하기 위해서는 시스템에서 발생한 상호작용에 대해 알아야할 필요가 있다. Fig. 6의 네 번째 실행 경로에 대한 상호작용 목록은 Table 2와 같다:

τ의 경우, 프로세스의 이동 전에 프로세스 간의 채널을 통해 프로세스가 동기 통신을 이룬 것을 의미한다. δ의 경우, 프로세스의 동기 통신이 끝난 후, 프로세스 간의 이동이 수행되었음을 의미한다. 상호작용 목록에 대한 GTS 생성 결과는 Fig. 7과 같다. GTS는 시스템의 프로세스들이 수행한 통신 및 이동을 2차원 공간에서 블록 형태로 표현한 그래프이다. Fig. 7의 각 블록은 프로세스의 이름 및 프로세스가 점유한 공간을 나타낸다. τ및 δ는 프로세스가 수행한 통신 및 이동이 표현된 것이다. Fig. 7의 GTS는 SAVE 도구를 사용하여 생성하였다.

4.4 검증(Verification)

PBC 예제의 모든 실행 경로에 대해 안전 요구사항이 만족하였는지 확인하기 위해, GTS-VL을 사용하여 안전 요구사항을 다시 명세할 필요가 있다. 위의 각 안전 요구사항을 GTS-VL을

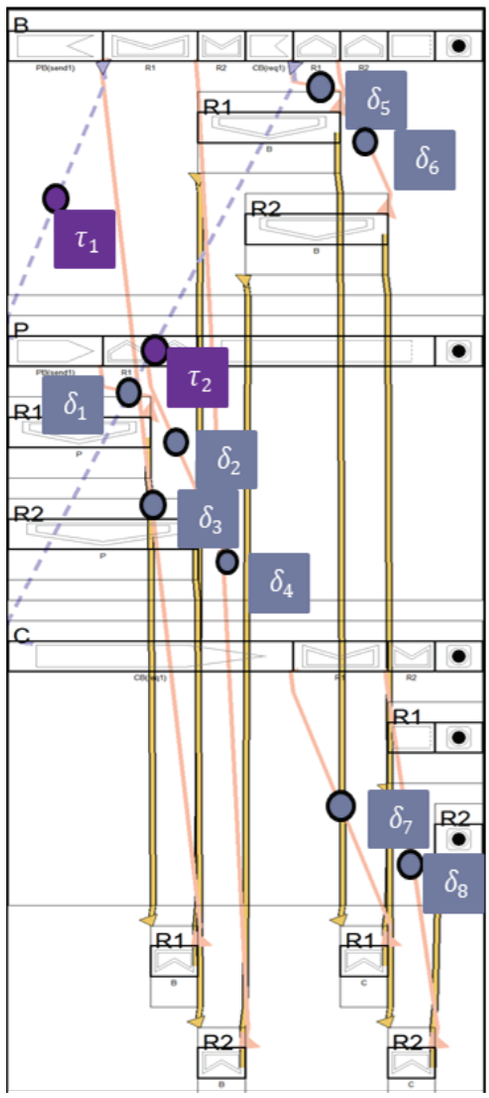


Fig. 7. GTS for The Fourth Execution Path

통해 명세하면 Table 3과 같이 네 개의 요구사항(*R1, R2, R3* 및 *R4*)이 도출된다. 각각의 요구사항에 대한 정의는 3.3.2절 시간 술어, 3.3.3절 공간 술어 및 3.3.4절 상호작용 술어에서 확인할 수 있다.

Table 4는 Fig. 7의 GTS를 기반으로 Table 3에서 명시한 네 개의 요구사항 검증 결과를 시각화한 것이다. Table 4의 시각화된 결과는 SAVE의 검증 도구에 의해 생성되었다. Fig. 3에서 정의한 각 술어에 대한 시각화된 연결선을 사용하여 문자 기반 요구사항 명세들이 GTS에서 시각화되며, 요구사항 검증 결과의 만족 여부는 연결선의 색상에 의해 표현된다.

각 요구사항들의 시각화를 하나의 GTS상에서 표현하면 Fig. 8과 같으며 다음과 같이 네 가지 요구사항의 만족 여부를 판별할 수 있다:

1) 최상단 공간의 *x*축은 각 요구사항의 시간 구간이 블록 형태로 표현된다.
2) 막대 형태로 표현된 요구사항 검증 결과는 연결선을 통

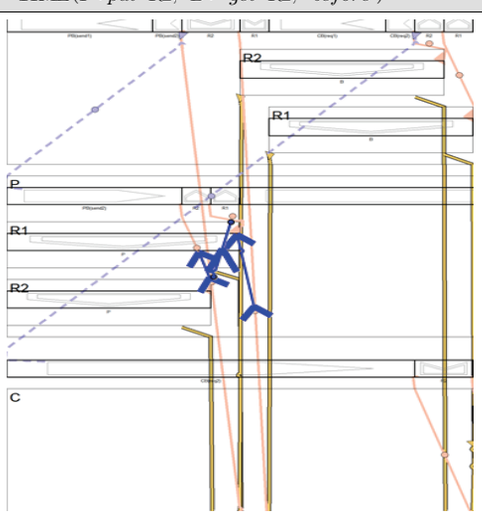
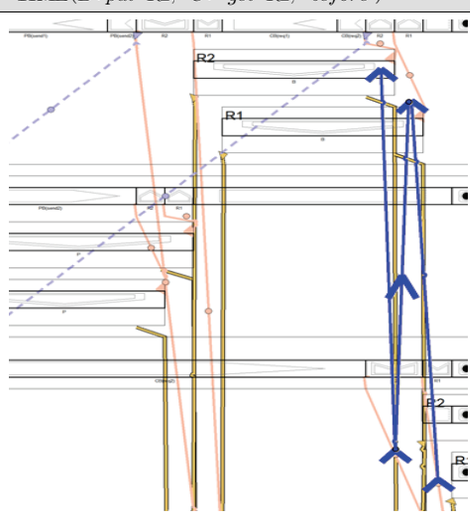
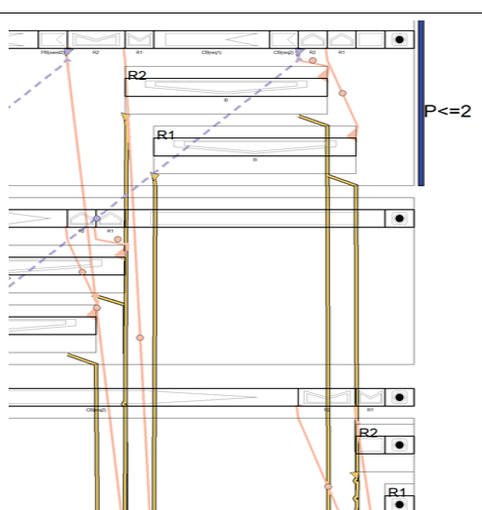
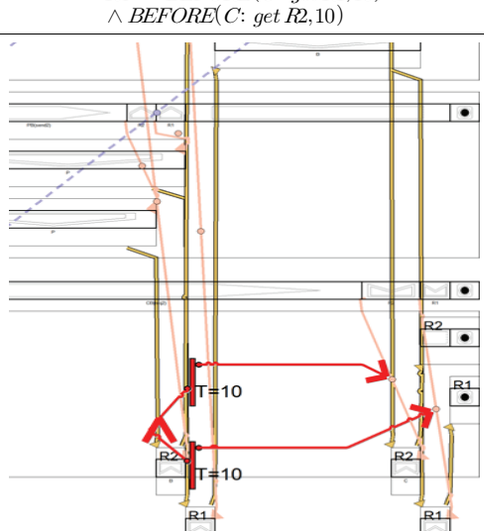
Table 3. Specification of Secure Requirements for PBC Example

Requirements	Semantic
$R1 ::=$ $TIME(P: put\ R1,$ $B: get\ R1, 'before') \wedge$ $TIME(P: put\ R2,$ $B: get\ R2, 'before')$	B gets $R1$ after R outs $R1$, and B gets $R2$ after P outs $R2$
$R2 ::=$ $TIME(B: put\ R1,$ $C: get\ R1, 'before') \wedge$ $TIME(B: put\ R2,$ $C: get\ R2, 'before')$	C gets $R1$ after B outs $R1$, and C gets $R2$ after B outs $R2$
$R3 ::= CAPA(B,2)$	The maximum capacity of B is 2
$R4 ::= BEFORE$ $(C: get\ R1, 10)$ $\wedge\ BEFORE$ $(C: get\ R2, 10)$	C gets $R1$ and $R2$ before 10 time units

해 GTS 상의 각 블록들과 연결되며, 각 연결선들은 명세한 술어의 종류에 따라 술어의 시각화가 이루어진다.

3) 요구사항의 만족 여부는 연결선 혹은 막대의 색을 통해 표현된다. 파란색은 연결선 혹은 막대는 요구사항이 만족했다는 의미이며, 빨간색 연결선 혹은 막대는 요구사항이 만족하지 못했다는 의미이다. 네 가지의 요구사항 명세 중 $R4$ 요구사항이 만족하기 못하였고, Fig. 8의 윗 부분의 $R1$, $R2$, $R3$ 및 $R4$ 요구사항 중에, $R4$ 요구사항 막대가 빨간색임을 확인할 수 있다. $R4$ 요구사항은 10 시간 단위 이내에 C가 자원1과 자원2를 받아야 한다는 것이며, 실행 분기 4에서는 10 시간 단위 이후에 $R1$ 및 $R2$를 받았기 때문에 요구사항을 만족하지 못했다. 각 안전 요구사항에 대한 의미는 Table 3에 요약되어 있다.

Table 4. Visualization of Verification Results for PBC Example

Requirement	$R1 ::= TIME(P: put\ R1,\ B: get\ R1, 'before') \wedge TIME(P: put\ R2,\ B: get\ R2, 'before')$	$R2 ::= TIME(B: put\ R1,\ C: get\ R1, 'before') \wedge TIME(B: put\ R2,\ C: get\ R2, 'before')$
Visualzation of Verification Result		
Require-ment	$R3 ::= CAPA(B,2)$	$R4 ::= BEFORE(C: get\ R1, 10) \wedge BEFORE(C: get\ R2, 10)$
Visualzation of Verification Result		

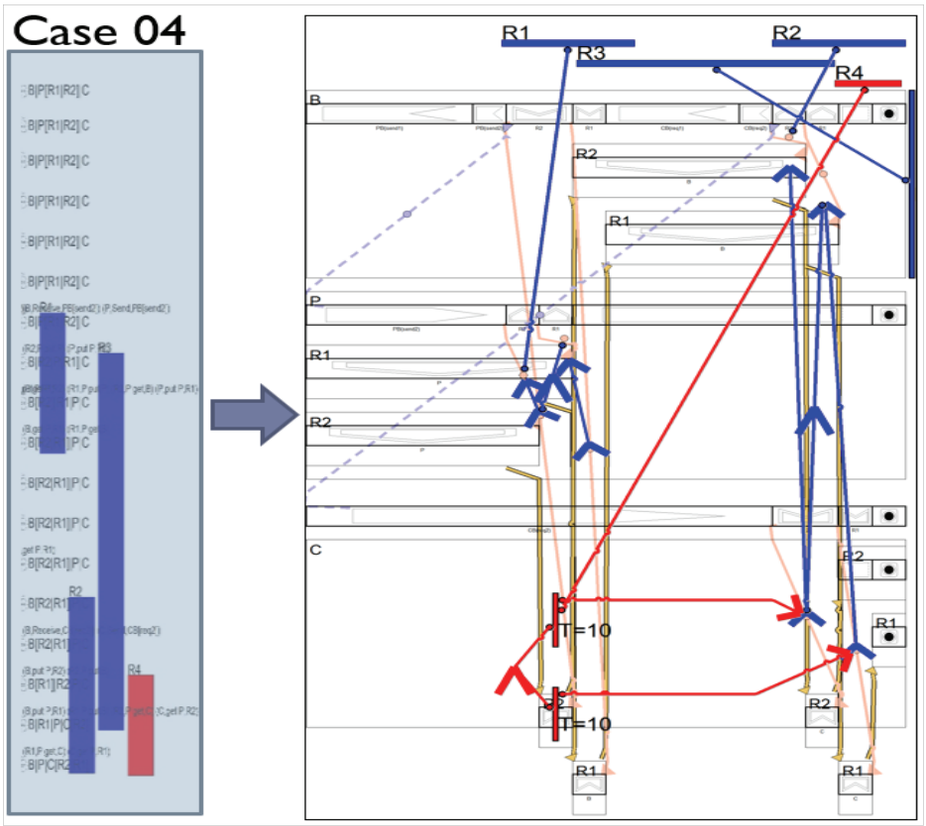


Fig. 8. Visualization of GTS-VL Verification Results: The Fourth Execution Path

이처럼 GTS-VL의 검증은 검증 결과의 시각화가 가능하다. GTS-VL의 검증 과정 및 결과를 시각화하지 않고 검증하려면, Table 2의 상호작용 목록에 명시된 프로세스 및 상호작용들을 Table 3의 안전 요구사항과 직접 대조해야 하므로 시각화 기반 요구사항 검증에 비해 검증 과정이 복잡하다. 하지만 GTS-VL은 검증 결과의 시각화 기능을 제공하기 때문에, 기존 문자 기반 논리에 비해 검증 결과를 쉽게 파악할 수 있다는 장점을 지니고 있다.

Fig. 9는 *PBC* 예제의 요구사항의 만족 여부가 실행모델 통해 요약되어 표현된 것이다. Fig. 9의 검증 결과는 SAVE 도구를 통해 일부 생성되었다. 시스템의 네 가지 실행 경로에 대해, *R1*부터 *R4*까지에 대한 요구사항 만족 여부가 막대의 길이 및 색상을 통해 표현된다. 막대의 길이는 요구사항에 해당되는 시간 구간을 나타낸다. 막대의 색은 다음을 의미한다:

1) 파란색: 요구사항이 만족함.
2) 빨간색: 요구사항이 만족하지 않음.

Fig. 9에서 실행모델의 각 네 가지 경로에 대해 R1부터 *R4*까지 요구사항이 만족되는 시간 구간이 막대 형태로 출력이 된다. 네 번째 실행 분기의 경우, *R4* 요구사항이 만족하지 않았기 때문에 빨간색 막대 형태로 표현되고, 나머지 요구사항들은 만족하였기 때문에 파란 막대 형태로 표현된다. Fig. 9의 각 case 별 요구사항 만족 여부를 표로 요약하면 Table 5와 같다:

Table 5. Summary of Verification of Secure Requirement for PBC

Req. \ Case	Case 01	Case 02	Case 03	Case 04
R1	O	O	O	O
R2	O	O	O	O
R3	O	O	O	O
R4	O	O	O	X

5. 비교분석

시간 및 공간을 명세하기 위한 다양한 언어들이 있다. 먼저 시간 논리는 대표적으로 LTL (*Linear Temporal Logic*), CTL (*Computational Tree Logic*), RTL (*Real-time Logic*) 등이 있다.

LTL은 하나의 시간 분기에서 분석 가능한 논리 언어이며, modal logic에 기초한 시간 논리이다. LTL은 always, eventually, release 등의 시간 연산자는 사용하여 프로세스가 어떠한 시점에 존재하는지 시간 연산자를 통해 명세할 수 있다.

CTL은 LTL을 확장하여 여러 시간 분기를 분석 가능한 논

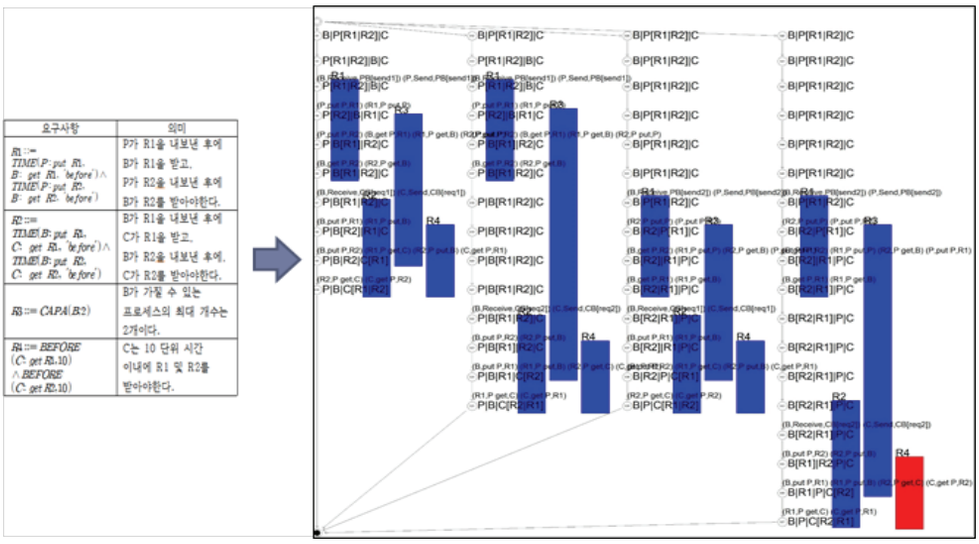


Fig. 9. Summary of GTS-VL Verification Results in Execution Model

리 언어이다. CTL은 all, exist, next, globally 등의 연산자를 사용하여 다양한 분기에 대한 시간 명세 및 분석을 수행할 수 있다.

RTL은 액션과 이벤트라는 요소를 기반으로 시스템의 요구사항을 명세 및 검증할 수 있는 언어이다. RTL은 시간, 정지, 상태 변수 전이, 외부 이벤트 및 전역 시간 등을 기반으로 시스템의 요구사항을 명세 및 분석할 수 있다.

기존의 시간 논리들은 시간에 기반하여 시스템이 수행하는 프로세스의 동기성이나 프로세스가 수행하는 행동의 선행 관계 등을 명세할 수 있지만, 스마트 IoT와 같이 프로세스의 분산성 및 이동성이 강조되는 시스템을 명세하기에는 프로세스의 이동성에 대한 표현이 직관적이지 못하다. 또한 시간 논리들은 문자에 기반하여 시스템을 명세 및 검증하기 때문에, 시스템의 명세 및 분석 결과를 시각화할 수 없다는 단점을 지니고 있다.

대표 공간 논리로는 RCC (*Region and Connection Calculus*), CRD (*Cardinal Direction Relations*) 등이 있다.

RCC는 두 공간 사이의 관계를 정의하여 두 공간에 대한 관계를 다루는 공간 논리이다. RCC를 사용하면 두 공간의 연결, 포함, 내접, 외접 등을 상세히 표현할 수 있는 장점을 지니고 있다. CRD는 공간의 좌표계에 기반한 공간 논리이며, 좌표에 따라 각 공간을 구분할 수 있다.

이러한 공간 논리들은 시스템을 명세 및 분석하기 위해 적합하지 않다. 그 이유는 공간 논리의 대부분은 단 두 개의 분리된 공간에 대해 논리에 기반한 명세가 특화되어 있기 때문에, 스마트 IoT와 같이 다수의 프로세스를 기존의 공간논리로 모두 표현하기에는 제한된다.

또한 기존의 공간 논리들은 시간을 표현할 수 없기 때문에, 프로세스의 상호작용에 기반한 이동성을 표현할 수 없다는 단점을 지니고 있으며, 공간이란 개념을 술어 정의에 기반한 추상화에 의해 시각화할 수 있지만, 논리 연결사는 시각화할 수 없는 등 공간 및 논리의 시각화가 제한되어있다.

GTS-VL은 GTS 상에서 표현된 블록들간의 관계를 다루는 1차술어논리이다. GTS-VL은 기존의 논리 언어들과 다르게 시간 및 공간을 동시에 분석 및 검증할 수 있다. 기존의 논리 언어들은 시간 혹은 공간 단일에 대한 관계에 대한 검증에 특화되어 있기 때문에 스마트 IoT의 분산성 및 이동성을 동시에 검증하기에는 제한적이며 직관적이지 못한다. 하지만 GTS-VL은 시간과 공간에 대한 검증이 모두 가능하기 때문에 스마트 IoT의 분산성 및 이동성을 검증하기에 적합하다. 또한 기존의 시간 및 공간 논리는 문자에 기반하여 요구사항을 분석 및 검증하기 때문에 분석 및 검증 과정에서 높은 복잡도를 유발하지만, GTS-VL은 검증 결과를 시각화하기 때문에 요구사항의 분석 및 검증에 대한 복잡도가 문자 기반 요구사항 분석 및 검증에 비해 상대적으로 낮은 편이다.

위와 같은 사항을 정량 분석하여 GTS-VL의 우수성을 입증하기 위해 문자화 복잡도(TC: Textual Complexity) 및 시각화 복잡도(VC: Visual Complexity)를 정의한다. TC 및 VC는 각각 문자에 기반한 요구사항을 분석할 때의 복잡도 및 시각화에 기반한 요구사항을 분석할 때의 복잡도를 의미한다. TC는 요구사항을 분석할 때, 각 요구사항에 대한 시스템의 상호작용에 대응하는 기호 및 이름의 합으로 정의된다[16]. VC는 검증 결과를 시각화하였을 때 나타나는 연결선으로 정의된다[17]. 복잡도 비교 분석 과정에 대한 예는 Fig. 10과 같다. Fig. 10은 PBC 예제의 R4 요구사항에 대한 문자기반 분석과 시각 기반 분석을 비교한 내용이다. Fig. 10에서 요구사항과 관련된 상호작용은 $\delta 7$ 및 $\delta 8$이며, 각 문자를 구성하는 술어 및 기호가 총 16개이므로 TC가 16임을 확인할 수 있다. VC의 경우, 요구사항의 분석 결과를 시각화한 결과로서 3개의 연결선이 생성되었으므로, VC는 총 3임을 확인할 수 있다. 그러므로 시각화에 기반하여 R4 요구사항을 분석하면, 문자 기반 요구사항 분석에 비해 총 13의 복잡도가 감소된다. PBC 예제의 모든 요구사항에 대한 TC 및 VC의 복잡

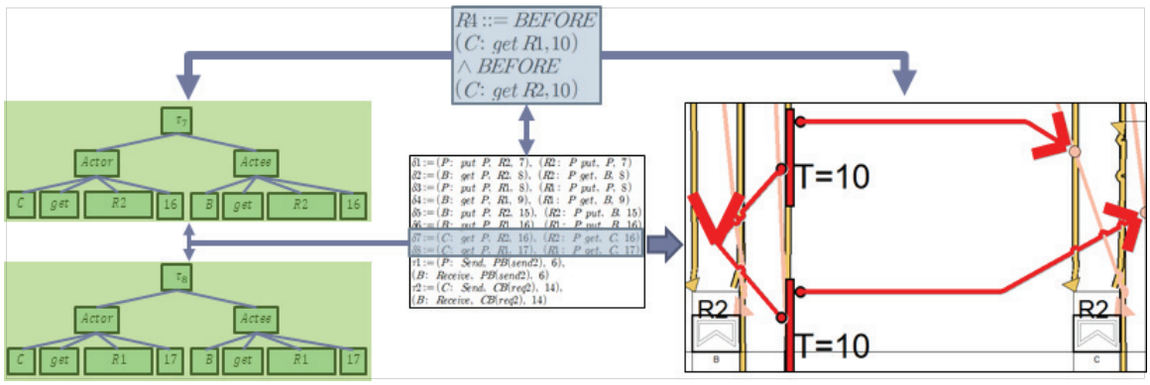


Fig 10. Comparative Analysis of Textual-based Requirement and Visual-based Requirements Analysis

Table 6. Comparative Analysis of Textual Complexity and Visual Complexity for PBC Requirements Analysis

Requirements	*TC* (Textual Complexity) = *Symbols* + *Names*	*VC* (Visual Complexity) = # *of Edges*	Complexity Reduction
$R1 ::= TIME(P: put\ R1,\ B:\ get\ R1,\ 'before') \wedge TIME(P: put\ R2,\ B:\ get\ R2,\ 'before')$ $R2 ::= TIME(B: put\ R1,\ C:\ get\ R1,\ 'before') \wedge TIME(B: put\ R2,\ C:\ get\ R2,\ 'before')$ $R3 ::= CAPA(B,2)$ $R4 ::= BEFORE(C:\ get\ R1,10) \wedge BEFORE(C:\ get\ R2,10)$	16+16+32+16 =82	5+5+3+5 =18	82-18 =64

도 감소를 요약한 결과는 Table 6과 같다. Table 6에서 TC의 복잡도는 총 82의 복잡도가 발생했고, VC의 경우 총 18의 복잡도가 발생했으므로 총 64의 복잡도가 감소된다.

6. SAVE

SAVE는 dTP-Calculus 및 GTS-VL로 시스템을 명세, 분석 및 검증하기 위해 저자가 개발한 도구이며, ADOxx Meta-Modeling Platform으로 개발되었다. ADOxx는 독일 베를린에 본부를 둔 OMiLAB국제연구소에서 개발한 메타-모델링 도구이다. ADOxx는 도구를 제작 및 사용하기 위한 모델링, 시뮬레이션, 매커니즘 및 알고리즘 등의 다양한 기능을 제공한다. ADOxx는 Fig. 11은 SAVE의 system architecture를 보여준다. SAVE는 ADOxx에 명세기, 분석기 및 검증기로 구성되어 있다.

6.1 명세기 (Specifier)

명세기(Specifier)는 dTP-Calculus에서 정의한 문법을 시각화하여 보여주는 모델이다. dTP-Calculus의 문법을 그래프로 나타낼 때, 다음과 같이 두 가지 모델을 사용하여 표현할 수 있다.

1) In-The-Large(*ITL*): *ITL*은 시스템에서 시스템을 구성하는 프로세스의 포함관계를 나타내는 모델이다. 프로세스는 다른 프로세스와 포함 또는 병렬 관계이며, 프로세스와 연결된 채널은 연결선과 이름을 통해 표현된다. Fig. 12는 SAVE 도구의 *ITL* 모델이다.
2) In-The-Small(*ITS*): ITS는 시스템을 구성하는 각 프로세스들이 수행하는 행동을 명세하는 모델이다. *ITS*의 큰 사각형 상자는 각 프로세스는 나타내고, 큰 사각형 상자 안의 다양한 아이콘들은 프로세스가 순차 실행하는 행동을 나타낸다. Fig. 13은 SAVE 도구의 ITS 모델이다.

6.2 분석기(Analyzer)

시스템의 명세가 완료되면 명세 모델을 기반으로 실행 경로를 분석한 결과인 실행모델을 생성할 수 있다. 실행모델은 명세 모델에서 발생할 수 있는 모든 경로를 분석하여 각 경로의 실행 결과를 tree 형태로 보여준다. 실행 결과가 정상이면 일반 아이콘으로 표기되며, 실행 중 deadlock이 발생하면 빨간색 아이콘으로 표기된다. Fig. 14는 SAVE의 실행모델이다.

6.3 검증기(Verifier)

검증기에서는 실행모델에서 생성된 실행 경로를 상세 분석하기 위해 GTS를 생성할 수 있다. GTS는 프로세스가 수행한 액션 및 이동을 2차원 시공간에서 표현한 모델이다. GTS에서 프로세스, 행동 및 상호작용은 모두 블록 형태로 표현된다. 또한 GTS-VL을 기반으로 GTS를 구성하는 블록들 간의 관계를 명세하여 시스템의 안전 요구사항을 검증할 수 있다. Fig. 15는 SAVE를 통해 생성된 GTS 모델이다.

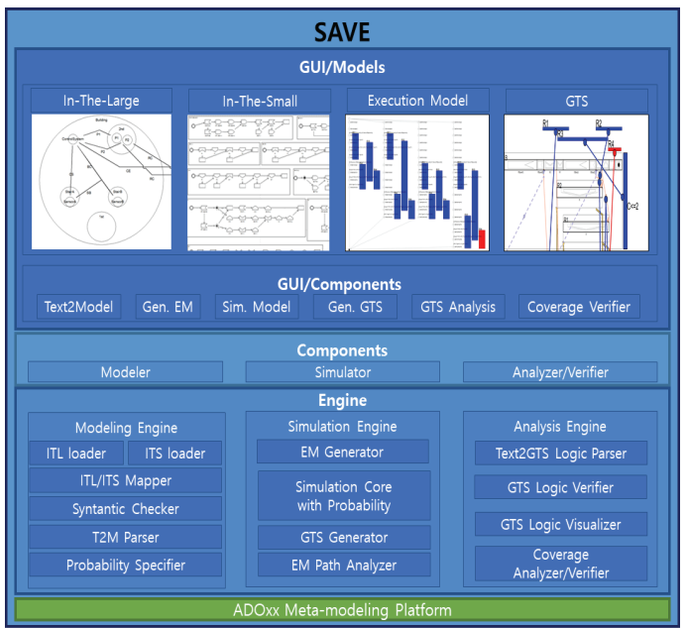


Fig 11. SAVE Architecture

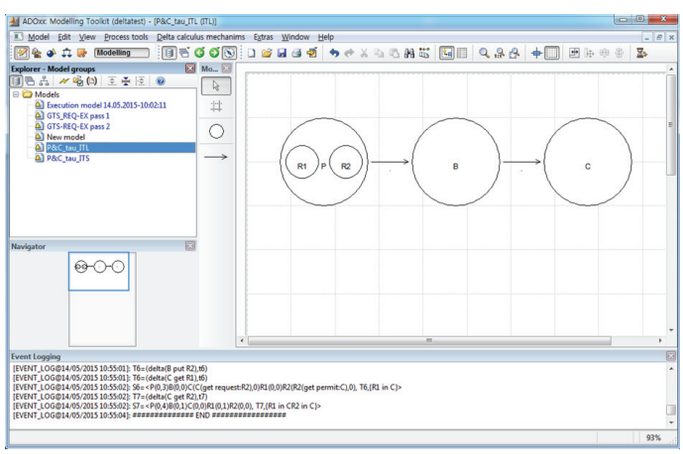


Fig. 12. Specification Tool of SAVE: In-The-Large

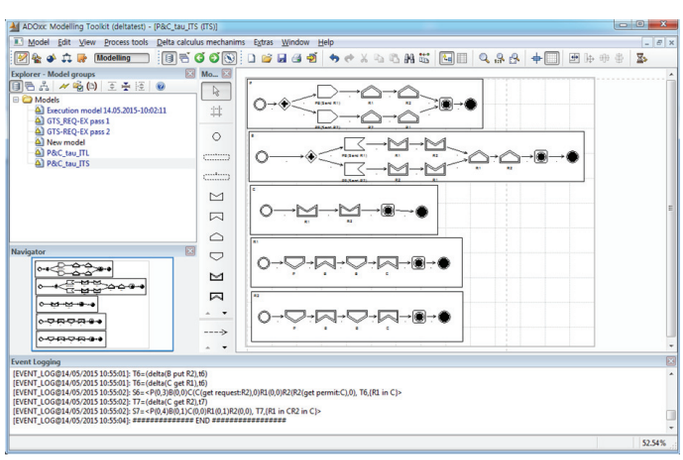


Fig. 13. Specification Tool of SAVE: In-The-Small

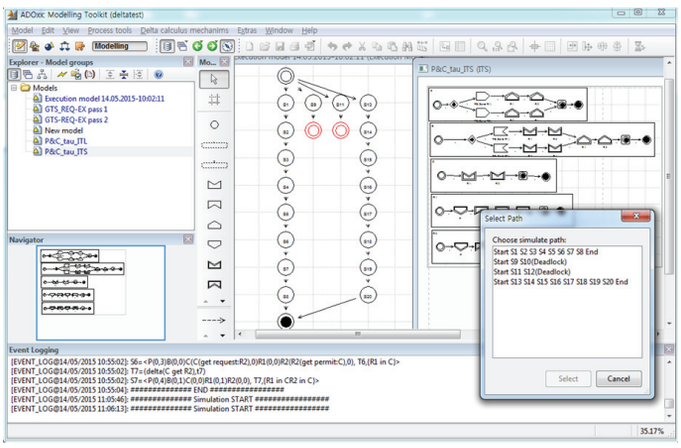


Fig. 14. Analysis Tool of SAVE

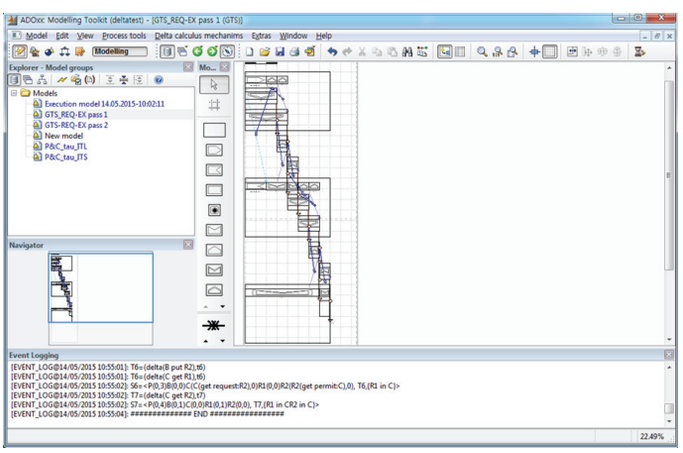


Fig. 15. Verification Tool of SAVE

7. 결론 및 향후 연구

4차 산업혁명의 주요 목표 중 하나는 스마트 IoT를 기반으로 한 자동화이다. 자동화를 위해 AI 및 빅데이터 등의 첨단 기술이 요구되기 때문에, 스마트 IoT의 안전성 및 보안성에 대한 요구사항이 비례적으로 강화되고 있다.

스마트 IoT를 구성하는 각 프로세스들은 시스템 내에서 분산성 및 이동성이라는 특징이 강조되기 때문에, 분산성 및 이동성의 특징에 기반한 시스템의 동작을 명세하고, 요구사항을 분석 및 검증할 수 있는 프로세스 대수 및 논리가 요구된다.

본 논문은 GTS-VL이라는 1차술어논리를 제시하였다. GTS-VL은 시스템의 구성하는 프로세스 간의 시공간 관계를 다루는 1차술어논리이며, 프로세스 대수인 dTP-Calculus로 명세한 시스템의 안전 요구사항을 분석 및 검증할 수 있다.

또한 요구사항 분석 결과를 시각화 할 수 있기 때문에, 스마트 IoT의 분산성 및 이동성이라는 특징을 직관적으로 표현할 수 있으며, 시각화 기반 요구사항 분석은 문자 기반 요구사항 분석에 비해 요구사항 분석의 복잡도가 줄어든다는 장점을 지니고 있다.

본 논문에서는 GTS-VL의 실용성을 입증하기 위해 dTP-Calculus 및 GTS-VL을 적용한 SAVE 도구를 사용하여 PBC 예제에 대한 안전 요구사항 분석을 수행하였다. PBC 예제의 모든 실행 경로를 도출하였고, 각 도출된 실행 경로에 대한 안전 요구사항을 검증하기 위해 GTS-VL에 기반한 검증을 수행 및 검증 결과를 시각화를 수행하였다.

안전 요구사항 검증 결과의 시각화에 대한 우수성을 입증하기 위해, 문자 기반 요구사항 분석의 복잡도와 시각화 기반 요구사항 분석의 복잡도를 정량적으로 비교하는 분석을 수행하였다. 비교 분석을 통해 시각화 기반 요구사항 분석이 문자 기반 요구사항 분석에 비해 복잡도가 크게 감소함을 입증하였다.

SAVE 도구는 저자가 개발한 시스템 명세, 분석 및 검증 도구이다. SAVE 도구는 ITL 및 ITS 명세기를 통해 시스템의 상태를 명세할 수 있으며, 분석기를 사용하여 명세한 시스템을 시뮬레이션 할 수 있다. 마지막으로 검증기를 기반으로 시스템의 안전 요구사항을 분석 및 검증 할 수 있다.

향후 연구는 GTS-VL의 효율성을 분석하기 위해 실제 스마트 IoT 사례에 대한 요구사항을 분석 및 검증할 것이다.

References

[1] K. Rob. "The real-time city? Big data and smart urbanism," *GeoJournal*, Vol.79, No.1, pp.1-14, 2014.

[2] Y. Choe and M. Lee, "Process model to predict non-deterministic behavior of IoT systems," *The 11th IFIP WG 8.1 working conference on the Practice of Enterprise Modelling (PoEM)*, pp.1-12, 2018.

[3] R. R. Smullyan, "First-order logic," Springer Science & Business Media, Vol.43, 2012.

[4] L. Cardelli and A. D. Gordon, "Mobile ambients," In *International Conference on Foundations of Software Science and Computation Structure*, Springer, pp.140-155, 1998.

[5] J. On, J. Choi, and M. Lee, "A study on scheduler based on CARDMI process algebra for automated control of emergency medical system," *Proceedings of the Korean Information Science Society Conference*, Korean Institute of Information Scientists and Engineers, pp.65-70, 2008.

[6] P. Coppin, J. Burton, and S. Hockema, "An attention based theory to explore affordances of textual and diagrammatic proofs," *International Conference on Theory and Application of Diagrams*, Springer, Berlin, Heidelberg, 2010.

[7] A. Shimojima and Y. Katagiri, "An eye-tracking study of exploitations of spatial constraints in diagrammatic reasoning," *International Conference on Theory and Application of Diagrams*, Springer, Berlin, Heidelberg, 2008.

[8] D. H. Ballard, M. M. Hayhoe, P. K. Pook, and R. P. N. Rao, "Deictic codes for the embodiment of cognition," *Behavioral and Brain Sciences*, Vol.20, No.4, pp.723-742, 1997.

[9] H. G. Fill and D. Karagiannis, "On the conceptua-lisation of modeling methods using the ADOxx meta modeling platform," *Enterprise Modeling and Information Systems Architectures (EMISAJ)*, Vol.8, pp.4-25, 2013.

[10] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronisation skeletons using branching time Temporal Logic," *Workshop on Logic of Programs. Springer*, pp.52-71, 1981.

[11] M. Huth and M. Ryan, "Logic in computer science: Modelling and reasoning about systems," Cambridge University Press, 2004.

[12] F. Jahanian and A. K. Mok, "Modechart: A specification language for real-time systems," *IEEE Transactions on Software Engineering*, Vol.20, pp.933-947, 1994.

[13] A. G. Cohn, B. Bennett, J. Gooday, and N. M. Gotts, "Qualitative spatial representation and reasoning with the region connection calculus," *GeoInformatica*, Vol.1, No.3, pp.275-316, 1993.

[14] A. U. Frank, "Qualitative spatial reasoning about distances and directions in geographic space," *Journal of Visual Languages and Computing*, Vol.3, No.4, pp.343-371, 1992.

[15] D. Ladret and M. Rueher, "VLP: A visual logic programming language," *Journal of Visual Languages & Computing*, Vol.2, No.2, pp.163-188, 1991.

[16] V. Gervasi and V. Ambriola, "Quantitative assessment of textual complexity," *Complexity in Language and Text*, pp.197-228, 2002.

[17] P. Coppin and S. Hockema, "A cognitive exploration of the 'non-visual' nature of geometric proofs," *Visual Languages and Logic*, pp.81-95, 2009.

이 성 현

https://orcid.org/0000-0002-4518-5067
e-mail : shlee1@jbnu.ac.kr
2015년 전북대학교 산업정보시스템공학 (학사)
2017년 전북대학교 컴퓨터공학부(석사)
2022년 전북대학교 컴퓨터공학부(박사)
관심분야 : 소프트웨어공학, 정형기법, 1차술어논리

이 문 근

https://orcid.org/0000-0003-2541-3066
e-mail : moonkun@jbnu.ac.kr
1989년 The Pennsylvania State University, Computer Science(학사)
1992년 The University of Pennsylvania, Computer and Information Science(석사)
1995년 The University of Pennsylvania, Computer and Information Science(박사)
1992년 ~ 1996년 미국, Computer Command and Control Company, Computer Scientist
1996년 ~ 1998년 전북대학교 컴퓨터과학과 전임강사
1998년 ~ 1999년 전북대학교 컴퓨터과학과 조교수
1999년 ~ 2007년 전북대학교 컴퓨터과학과 부교수
2007년 ~ 현 재 전북대학교 컴퓨터공학부 교수
관심분야 : 정형기법, 소프트웨어 재/역공학, 실시간 시스템, 운영체제, 형식언어, 병렬함수언어, 컴파일러 등