A Dual Method to Model IoT Systems

Sunghyeon Lee, Yeongbok Choe, Moonkun Lee*
The Chonbuk National University
Jeonju
Republic of Korea
moonkun@jbnu.ac.kr

Abstract— This paper presents a dual method: 1) to specify both dynamic properties, such as operational requirements, and static properties, such as safety requirements, of IoT systems, whose services are distributed over a geographical space and are mobile over the space in time, and 2) verify the validation of the static properties over the dynamic properties. Firstly, the dynamic properties are specified with a process algebra, called δ-Calculus, and the static properties are specified a first-order logic, called GTS Logic. Secondly, once specifications are done, the static properties are verified for its validity over the dynamic properties. For example, safety requirements are verified to see its validity over operational requirements by checking whether or not the safety requirements are satisfied for all the possible simulation cases of the operational requirements. In order to demonstrate the feasibility of the method, a tool, namely SAVE, is developed on a meta-modeling platform, namely ADOxx. The method and the tool can be considered one of the most innovative approaches to model the IoT systems.

***Keywords*—** Dual Method; Visualization; SAVE; δ-Calculus; GTS Logic; Specification; Verification; ADOxx

I. INTRODUCTION

There are strong needs for formal methods to model IoT systems [1]. However the most of methods provide capability to model either static properties or dynamic properties, not both. Therefore it is desirable to model both dynamic and static properties, and to verify the validity of the static properties over the dynamic properties for IoT [2].

This paper presents a new method to specify IoT systems with both dynamic properties, such as, operational requirements, and static properties, such as, safety requirements, and to verify the validity of the static properties over the dynamic properties, such as, the validity of safety requirements over the operational requirements, as follows:

1) Specifications:
   i) Operational requirements as dynamic properties: These requirements are specified with a process algebra, called *δ-Calculus* [3]. The calculus is designed to specify DMRTS, applicable to IoT systems.
   ii) Safety requirements as static properties: These requirements are specified with a first order logic, called *GTS*(*Geo-Temporal Space*) *Logic* [4]. The logic is designed to reason geo-temporal properties of processes and their interactions on a specific geographical space for DMRTS, applicable to IoT.
2) Verification: In order to verify the safety requirements for the dynamic requirements, it is necessary to generate all the possible execution cases for simulation. It is accomplished as follows:
   i) Simulation:
      i. Execution model: A visual execution model is generated for the operational specification. It contains all the possible cases for execution.
      ii. Simulation: It simulates each individual case in the execution model. It represents the results of the simulation in a GTS block diagram, which will be input to Verifier for verification.
   ii) Verification: It verifies visually on the GTS block diagrams for the safety requirements in GTS Logic. The results of the verification will be displayed directly and visually on the diagrams.

The approach will be demonstrated in a tool, called, SAVE (*Specification, Analysis, Verification and Evaluation*) [4], which is developed on the ADOxx meta-modeling platform [5], with an example, for the feasibility of the method.

The method can be considered as one of the innovative methods to model IoT systems to specify both static and dynamic properties of the systems, as well as to verify the validity of static properties over the dynamic properties.

The organization of the paper is as follows. Section 2, 3 and 4 describes δ-Calculus, Execution Model and GTS Logic, respectively. Section 5 demonstrates the method on SAVE with a PBC (Producer-Buffer-Consumer) example. Finally conclusions will be made with future research.

II. OPERATIONAL SPECIFICATION

A. δ-Calculus

δ-Calculus is a process algebra to model the behavior of processes by defining distribution of processes on a geographical space and their actions, especially movements in time and with priority.

[Definition 1] (δ-Calculus)

δ-Calculus is defined as a tuple $\delta = (Syn, Sem, Law)$, where Syn, Sem and Law are defined in Def. 2, 3 and 4, respectively.

* Corresponding Author

[Definition 2] (*Syn*: Syntax of δ-Calculus)

$\mathrm{P} ::= \mathrm{nil}$	//Inaction
$\mid \mathrm{A}$	//Action
$\mid \mathrm{P}_{(\mathrm{n})}$	//Priority
$\mid \mathrm{P}[\mathrm{Q}]$	//Nesting
$\mid \mathrm{P}\langle r_t \rangle$	//Channel
$\mid \mathrm{P} + \mathrm{Q}$	//Choice
$\mid \mathrm{P} \mid \mathrm{Q}$	//Parallel
$\mid \mathrm{A} \cdot \mathrm{P}$	//Sequence
$\mathrm{A} ::= \emptyset$	//Empty Action
$\mid \mathrm{r_t}(\overline{msg})$	//Send
$\mid \mathrm{r_t}(msg)$	//Receive
$\mid \mathrm{M}$	//Movement
$\mathrm{M} ::= \mathrm{m}_\mathrm{t}^\mathrm{p}(k)\ P$	//Request
$\mid \mathrm{P\ m(k)}_\mathrm{t}^\mathrm{p}$	//Permission
$\mathrm{m} ::= \mathrm{in}$	//In Movement
$\mid \mathrm{out}$	//Out Movement
$\mid \mathrm{get}$	//Get Movement
$\mid \mathrm{put}$	//Put Movement

[Definition 3] (*Sem*: Semantics of δ-Calculus)

1) Sem_τ: Communicaiton

Action	$\dfrac{-}{\mathrm{r(a)} \cdot \mathrm{P} \xrightarrow{\mathrm{r(a)}} \mathrm{P}}$
ChoiceL	$\dfrac{\mathrm{P} \xrightarrow{A} P'}{\mathrm{P} + \mathrm{Q} \xrightarrow{\mathrm{A}} \mathrm{P}'}$
ChoiceR	$\dfrac{\mathrm{Q} \xrightarrow{A} Q'}{\mathrm{P} + \mathrm{Q} \xrightarrow{\mathrm{A}} \mathrm{Q}'}$
ParlL	$\dfrac{\mathrm{P} \xrightarrow{A} P'}{\mathrm{P} \mid \mathrm{Q} \xrightarrow{\mathrm{A}} P' \mid Q}$
ParlR	$\dfrac{\mathrm{Q} \xrightarrow{A} Q'}{\mathrm{P} \mid \mathrm{Q} \xrightarrow{\mathrm{A}} P \mid Q'}$
ParCom	$\dfrac{\mathrm{P} \xrightarrow{A} P', \mathrm{Q} \xrightarrow{\bar{A}} Q'}{\mathrm{P} \mid \mathrm{Q} \xrightarrow{\tau} P' \mid Q'}$
NestO	$\dfrac{\mathrm{P} \xrightarrow{A} P'}{\mathrm{P}[\mathrm{Q}] \xrightarrow{\mathrm{A}} P'[Q]}$
NestI	$\dfrac{\mathrm{Q} \xrightarrow{A} Q'}{\mathrm{P}[\mathrm{Q}] \xrightarrow{\mathrm{A}} P[Q']}$
NestCom	$\dfrac{\mathrm{P} \xrightarrow{A} P', \mathrm{Q} \xrightarrow{\bar{A}} Q'}{\mathrm{P}[\mathrm{Q}] \xrightarrow{\tau} P'[Q']}$

2) Sem_δ: Movements

In	$\dfrac{\mathrm{P} \xrightarrow{\mathrm{in_t(k)\ Q}} \mathrm{P}', \mathrm{Q} \xrightarrow{\mathrm{P\ in(k)}} \mathrm{Q}'}{\mathrm{P} \parallel \mathrm{Q} \xrightarrow{\delta} \mathrm{Q}'[\mathrm{P}']}$
Out	$\dfrac{\mathrm{P} \xrightarrow{\mathrm{out_t(k)\ Q}} \mathrm{P}', \mathrm{Q} \xrightarrow{\mathrm{P\ out(k)}} \mathrm{Q}'}{\mathrm{Q}[\mathrm{P}] \xrightarrow{\delta} \mathrm{P}' \parallel \mathrm{Q}'}$
Get	$\dfrac{\mathrm{P} \xrightarrow{\mathrm{get_t(k)\ Q}} \mathrm{P}', \mathrm{Q} \xrightarrow{\mathrm{P\ get(k)}} \mathrm{Q}'}{\mathrm{P} \parallel \mathrm{Q} \xrightarrow{\delta} \mathrm{P}'[\mathrm{Q}']}$
Put	$\dfrac{\mathrm{P} \xrightarrow{\mathrm{put_t(k)\ Q}} \mathrm{P}', \mathrm{Q} \xrightarrow{\mathrm{P\ put(k)}} \mathrm{Q}'}{\mathrm{P}[\mathrm{Q}] \xrightarrow{\delta} \mathrm{P}' \parallel \mathrm{Q}'}$
InP	$\dfrac{\mathrm{P}_{(\mathrm{n})} \xrightarrow{\mathrm{in_t^p(k)\ Q_{(m)}}} \mathrm{P}'_{(\mathrm{n})}}{\mathrm{P}_{(\mathrm{n})} \parallel \mathrm{Q}_{(\mathrm{m})} \xrightarrow{\delta} \mathrm{Q}_{(\mathrm{m})}[\mathrm{P}_{(\mathrm{n})}']} (\mathrm{n} \geq \mathrm{m})$
OutP	$\dfrac{\mathrm{P}_{(\mathrm{n})} \xrightarrow{\mathrm{out_t^p(k)\ Q_{(m)}}} \mathrm{P}'_{(\mathrm{n})}}{\mathrm{Q}_{(\mathrm{m})}[\mathrm{P}_{(\mathrm{n})}] \xrightarrow{\delta} \mathrm{P}'_{(\mathrm{n})} \parallel \mathrm{Q}_{(\mathrm{m})}} (\mathrm{n} \geq \mathrm{m})$
GetP	$\dfrac{\mathrm{P}_{(\mathrm{n})} \xrightarrow{\mathrm{get_t^p(k)\ Q_{(m)}}} \mathrm{P}'_{(\mathrm{n})}}{\mathrm{P}_{(\mathrm{n})} \parallel \mathrm{Q}_{(\mathrm{m})} \xrightarrow{\delta} \mathrm{P}'_{(\mathrm{n})}[\mathrm{Q}_{(\mathrm{m})}]} (\mathrm{n} \geq \mathrm{m})$
PutP	$\dfrac{\mathrm{P}_{(\mathrm{n})} \xrightarrow{\mathrm{put_t^p(k)\ Q_{(m)}}} \mathrm{P}'_{(\mathrm{n})}}{\mathrm{P}_{(\mathrm{n})}[\mathrm{Q}_{(\mathrm{m})}] \xrightarrow{\delta} \mathrm{P}'_{(\mathrm{n})} \parallel \mathrm{Q}_{(\mathrm{m})}} (\mathrm{n} \geq \mathrm{m})$
InN	$\dfrac{\mathrm{P} \xrightarrow{\mathrm{in_t(k)\ Q}} \mathrm{P}', \mathrm{Q} \xrightarrow{\mathrm{P\ in(k)}} \mathrm{Q}'}{\mathrm{P} \parallel \mathrm{Q}[\mathrm{R}] \xrightarrow{\delta} \mathrm{Q}'[\mathrm{P}' \parallel \mathrm{R}]}$
GetN	$\dfrac{\mathrm{P} \xrightarrow{\mathrm{get_t(k)\ Q}} \mathrm{P}', \mathrm{Q} \xrightarrow{\mathrm{P\ get(k)}} \mathrm{Q}'}{\mathrm{P}[\mathrm{R}] \parallel \mathrm{Q} \xrightarrow{\delta} \mathrm{P}'[\mathrm{R} \parallel \mathrm{Q}']}$

[Definition 4] (*Law*: Algebraic Laws)

$\mathrm{P} + \mathrm{P} = \mathrm{P}$	Choice(1)
$\mathrm{P} + \mathrm{Q} = \mathrm{Q} + \mathrm{P}$	Choice(2)
$(\mathrm{P} + \mathrm{Q}) + \mathrm{R} = \mathrm{P} + (\mathrm{Q} + \mathrm{R})$	Choice(3)
$\mathrm{P} \parallel \emptyset = \mathrm{P}$	Parallel(1)
$\mathrm{P} \parallel \mathrm{Q} = \mathrm{Q} \parallel \mathrm{P}$	Parallel(2)
$(\mathrm{P} \parallel \mathrm{Q}) \parallel \mathrm{R} = \mathrm{P} \parallel (\mathrm{Q} \parallel \mathrm{R})$	Parallel(3)
$\mathrm{P}[\emptyset] = \mathrm{P}$	Nesting(1)
$\mathrm{R}[\mathrm{P}] + \mathrm{R}[\mathrm{Q}] = \mathrm{R}[\mathrm{P} + \mathrm{Q}]$	Nesting(2)
$\mathrm{P} \parallel (\mathrm{Q} + \mathrm{R}) = (\mathrm{P} \parallel \mathrm{Q}) + (\mathrm{P} \parallel \mathrm{R})$	Distributive(1)
$(\mathrm{a}_1 + \mathrm{a}_2).\mathrm{P} = \mathrm{a}_1.\mathrm{P} + \mathrm{a}_2.\mathrm{P}$	Distributive(2)

[Example 1] (*PBC* Example)

A *Producer-Buffer-Consumer*(PBC) Example is shown in Fig. 1. There are five parallel processes, where *R*1 and *R*2 are in *P*. Here *P* sends non-deterministically the *send R*1 or the *send R*2 message to *B* through the *PB* channel to inform *B* to *put R*1 or *R*2 first, and *put*s *R*1 or *R*2 out of its space in order. After receiving the message, *B get*s *R*1 or *R*2 nondeterministically, then *B put*s *R*1 and *R*2 out of its space in order. Finally *C get*s *R*1 and *R*2 in order.

$$PBC := P[R1,R2] \mid B \mid C;$$
$$P := \left(PB(\overline{send\ R1})\bullet put\ R1\bullet put\ R2 + PB(\overline{send\ R2})\bullet put\ R2\bullet put\ R1\right)\bullet exit;$$
$$B := \left(PB(send\ R1)\bullet get\ R1\bullet get\ R2 + PB(send\ R2)\bullet get\ R2\bullet get\ R1\right)\bullet$$
$$put\ R1\bullet put\ R2\bullet exit;$$
$$C := get\ R1\bullet get\ R2\bullet exit;$$
$$R1 := P\ put\bullet B\ get\bullet B\ put\bullet C\ get\bullet exit;$$
$$R2 := P\ put\bullet B\ get\bullet B\ put\bullet C\ get\bullet exit;$$

Fig. 1 PBC Example in δ-Calculus

B. Execution Model for δ-Calculus

The execution model for δ-Calculus is based on the notion of system state and its transition. Since the system consists of processes, its state is defined as a set of states of its processes with the two additional state variables: a set of inclusion relations among processes and a global clock. In order to control temporal synchrony of actions, two types of clocks are defined: 1) the global clock for the system and 2) the local clocks for each processes. The former is named as the global clock (*T*) and the latter is named as local clocks (t_i) for each process P_i.

[Definition 4] (*Process State*)

Process state is $p_i = (\bullet_i, t_i)$, where $\bullet_i$ is the position just after a_i in the sequence of actions in *P* and t_i is the time of the local clock. There are two special states: *start* and *final*.

[Definition 5] (*Process State Transition*)

Process state transition is $p_{j-1} \xrightarrow{a_j^t = (a_j, t_j)} p_j$, where a transition occurs from p_{j-1} to p_j by an action a_j, which takes the time t_i.

The transition is based on the rules defined in Def. 3. For example, if $p_0 = (0,0)$ and the first action of sending a message on the channel *c* in *P* occurs in the time 2, then $p_1 = (1,2)$ by $p_0 \xrightarrow{c(\overline{m}),2} p_1$.

[Definition 6] (*Execution Model for Process*)

Execution model for processes in δ-calculus is $p_0 \xrightarrow{a_1^{t1}} p_1 \xrightarrow{a_2^{t2}} p_2 \longrightarrow \cdots \longrightarrow p_f$, where each p_i is a process state, which is transited to the next state after performing an action a_i in the time t_i. There are two special states, s_0 and s_f, for the *start* and *final* states, respectively.

It is possible not only to have multiple transitions from one process state, but also to one process state. And it is possible not only to have no *final* state, but also to have multiple *final* states. However there should be a single *start* state.

[Definition 7] (*System State*)

System state is $s_i = \left((p_{1,i}, p_{2,i}, \cdots, p_{n,i}), I_i, C_i, T_i\right)$, where each $p_{j,i}$, I_i, C_i and t_i represent the state of each process P_i, a set of the inclusion relations among processes, a set of channels and the global time, respectively.

[Definition 8] (*System State Transition*)

System state transition between one system state to another state is $s_j \xrightarrow{i_j^t} s_{j+1}$, where $i_j = \left((P_a : a, P_{\bar{a}} : \bar{a}), t_j\right)$. Here s_j and s_{j+1} are system states, i_j^t is a synchronous interaction between processes P_a and $P_{\bar{a}}$ with the action a of P_a and the action $\bar{a}$ of $P_{\bar{a}}$ in the time t_j. The transition is based on the rules defined in the semantics of δ-calculus.

[Definition 9] (*System State Transition*)

System execution model for δ-calculus is the labelled transition system: $s_0 \xrightarrow{i_1} s_1 \xrightarrow{i_2} s_2 \longrightarrow \cdots \longrightarrow s_f$. Each s_j represents a system state, and i_k does a system state transition. There are two special states, s_0 and s_f, for *first* and *final* states, respectively.

It is possible not only to have multiple transitions from one system state, but also to one system state. And it is possible not only to have no *final* state, but also to have multiple *final* states. However there should be a single *start* state.

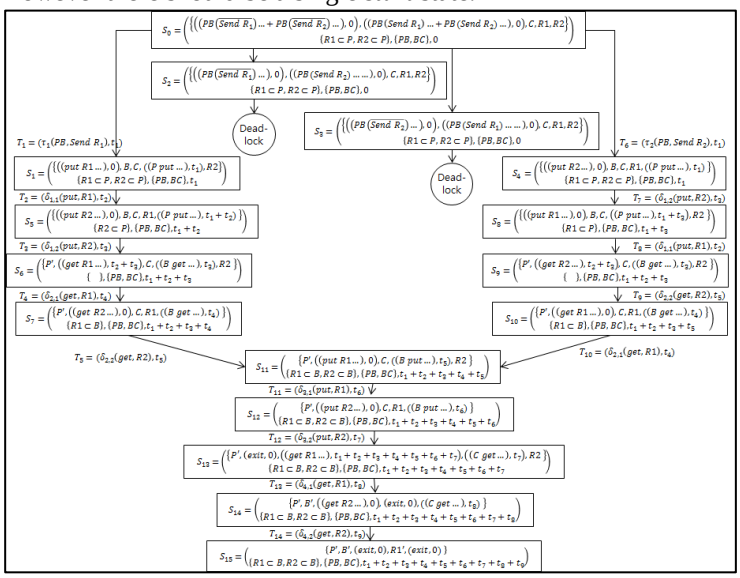

Fig. 2 Execution Model for PBC Example

[Example 2] (Execution Model for PBC Example)

Execution Model for PBC example is shown in Fig 2.

III. NON-FUNCTIONAL SPECIFICATION

A. GTS

Geo-Temporal Space (GTS) is the two-dimensional space for processes running over time. It can be represented as the composition of one-dimensional process inclusion relations on a geo-graphical area and one-dimensional temporal space.

[Definition 10] (*Temporal Dimension* (DD))

Time space is represented as 1-dimensional space of integers.

[Example 2]

TS for 100 time units can be represented as $TS = [0, 1, 2, \dots, 100]$.

[Definition 11] (*Geographical Dimension* (PD))

The logical boundary of processes in a system on a geographical area can be represented linearly in 1-dimensional space as follows:

1) Base case: Process *P* can be represented with a pair of logical lower (*l*) and upper (*u*) boundaries: $P = [p_l, p_u]$, where $p_l < p_u$.
2) Induction case: There are two possible geographical relations between any two processes *P* and *Q*:
   i) Parallel ($P||Q$): $[p_l, p_u]||[q_l, q_u] = [p_l, p_u, q_l, q_u] \vee [q_l, q_u, p_l, p_u]$
   ii) Inclusion ($P[Q] \vee Q[P]$): $[p_l, [q_l, q_u], p_u] \vee [q_l, [p_l, p_u], q_u] = [p_l, q_l, q_u, p_u] \vee [q_l, p_l, p_u, q_u]$

Note that *P* and *Q* can be inductively defined.

[Example 3]

The PBC System that consists of 3 processes Producer with 2 Resources, Buffer and Consumer can be represented as $\text{PBC}[P[R_1, R_2], B, C] = [pbc_l, p_l, r1_l, r1_u, r2_l, r2_u, p_u, b_l, b, c_l, c_u, pbc_u]$.

[Proposition 1] (GS *No Partiality* (GS-PG) Law)

There is no partial geographical intersection among processes.

(Proof):

By the definition of δ-Calculus, for any process *P* and *Q*, there are three possible inclusion relations: $P[Q]$, $Q[P]$ or $P||Q$.

[Proposition 2] (*GS Linear Order* (GS-LO) Law)

All logical boundaries of the processes in a system can be linearly ordered in PS.

(Proof):

By Def. 11, all the possible compositions of the logical boundaries of a system are ordered linearly as follows:

1) Base case: $P = [p_l, p_u]$, where $p_l < p_u$.
2) Induction case:
   i) Parallel ($P||Q$): $[p_l, p_u]||[q_l, q_u] = [p_l, p_u, q_l, q_u] \vee [q_l, q_u, p_l, p_u]$, where $p_l < p_u < q_l < q_u$ for $[p_l, p_u, q_l, q_u]$, and $q_l < q_u < p_l < p_u$ for $[q_l, q_u, p_l, p_u]$.
   ii) Inclusion ($P[Q] \vee Q[P]$): $[p_l, [q_l, q_u], p_u] \vee [q_l, [p_l, p_u], q_u] = [p_l, q_l, q_u, p_u] \vee [q_l, p_l, p_u, q_u]$, where $p_l < q_l < q_u < p_u$ for $[p_l, q_l, q_u, p_u]$, and $q_l < p_l < p_u < q_u$ for $[q_l, p_l, p_u, q_u]$.

Note that *P* and *Q* can be inductively defined.

[Definition 12] (Interval)

Interval $I = [a, b]$ is defined as a distance between two discrete values, a and b, $a < b$, in an ordered discrete value domain, such as Integer or Alphabets. For GTS, there are two types of intervals:

1) Time Interval (TI).
2) Geographical Interval (GI).

[Definition 13] (Relations/Operations for Intervals)

Interval relations and operations are defined as follows for $I_1 = [a_1, b_1]$, $I_2 = [a_2, b_2]$:

1) Relations
   i) $=$ (Equal): $I_1 = I_2$, if $a_1 = a_2 \wedge b_1 = b_2$.
   ii) $<$ (No-overlap): $I_1 < I_2$, if $b_1 < a_2$.
   iii) $\leq$ (Contingent): $I_1 \leq I_2$, if $b_1 = a_2$.
   iv) $\sqsubset$ (In-overlap): $I_1 \sqsubset I_2$, if $a_1 > a_2 \wedge b_1 < b_2$.
   v) $\sqsubseteq$ (Overlap): $I_1 \sqsubseteq I_2$, if $a_1 < a_2 \wedge b_1 < b_2 \wedge a_2 < b_1$.
2) Operations:
   i) $\sqcup$ (Union):
      • $I_1 \sqcup I_2 = [a_1, b_1, a_2, b_2]$, if $I_1 < I_2 \vee I_1 \leq I_2$;
        $= I_2$, if $I_1 \sqsubset I_2$;
        $= I_3 = [a_1, b_2]$, if $I_1 \sqsubseteq I_2$.
   ii) $\sqcap$ (Intersection):
      • $I_1 \sqcap I_2 = \emptyset$, if $I_1 < I_2 \vee I_1 \leq I_2$;
        $= I_1$, if $I_1 \sqsubset I_2$;
        $= I_3 = [a_2, b_1]$, if $I_1 \sqsubseteq I_2$.
   iii) ^{C} (Complement) :
      • $I^C = [a, b]^C$
        $= [d_l, \cdots, a, b, \cdots, d_u] - [a, b]$
        $= [d_l, \cdots, \cdots, d_u]$,
        where $P \in D = [d_l, d_u]$.
   iv) $-$ (Difference) :
      i. $I_1 < I_2$ (No-overlap): $I_1 - I_2 = I_1$.
      ii. $I_1 \leq I_2$ (Contingent): $I_1 - I_2 = I_1$.
      iii. $I_1 \sqsubset I_2$ (In-overlap):
         • $I_{1,2} - I_1 = [a_2, [a_1, b_1], b_2] - [a_1, b_1]$
           $= [a_2, b_2] = I_2$.
      iv. $I_1 \sqsubseteq I_2$ (Overlap): $I_{1,3} - I_{3,2} = [a_1, [a_3, b_1], b_1] - [a_2, [a_2, b_3], b_2] = [a_1, a_3]$, where $I_3 = I_1 \sqcap I_2$.

[Definition 14] (Functions for Intervals)

Interval functions are defined as follows:

1) $begin(TI) = t_b$ if $TI = [t_b, t_e]$.
2) $end(TI) = t_e$ if $TI = [t_b, t_e]$.
3) $length(TI) = t_e - t_b$ if $TI = [t_b, t_e]$.
4) $lower(GI) = g_l$ if $TI = [g_l, g_u]$.
5) $upper(GI) = g_u$ if $TI = [g_l, g_u]$.

[Proposition 3] (GT Relations and Operations)

1) Base case: $P = [p_l, p_u]$, where $p_l < p_u$.
   i) Relations: $P = P$
   ii) Functions:
      i. $\sqcup$ (Union): $P \sqcup P = P$
      ii. $\sqcap$ (Intersection): $P \sqcap P = P$
      iii. ^{C}(Complement):
         $P^C = [s_l, \cdots, p_l, p_u, \cdots, s_u] - [p_l, p_u]$

$= [s_l, \cdots, \cdots, s_u]$, where $P \in S$.

iv. $-$ (Difference) : $P - P = [\,] = \emptyset$

2) Induction case (Parallel ($P||Q$)):
$P||Q = [p_l, p_u]|||[q_l, q_u] = [p_l, p_u, q_l, q_u] \vee [q_l, q_u, p_l, p_u]$

i) Relations: $P < Q \vee P < Q$

ii) Functions:

i. $\sqcup$(Union): $P \sqcup Q = [p_l, p_u, q_l, q_u] \vee [q_l, q_u, p_l, p_u]$

ii. $\sqcap$ (Intersection): $P \sqcap Q = \emptyset$

iii. $-$ (Difference) : $P - Q = P$

3) Induction case (Inclusion ($P[Q]$)): $P[Q] = [p_l, [q_l, q_u], p_u] = [p_l, q_l, q_u, p_u]$

i) Relations: $Q \sqsubset P$

ii) Functions:

i. $\sqcup$ (Union): $P[Q] \sqcup Q = P$

ii. $\sqcap$ (Intersection): $P[Q] \sqcap Q = Q$

iii. $-$ (Difference) : $P[Q] - Q = P$

4) Induction case (Inclusion ($Q[P]$)): Similar to Case 3).

(Proof):

By Def. 13.

[Definition 15] (GTS (Geo-Temporal Space))

GTS is defined as $S = \langle G, T \rangle$, where $G = [g_u, g_l]$, $T = [t_b, t_e]$. It implies that *GTS* is the composition of the geo-space *G* of the size $[g_u, g_l]$, where $g_u < g_l$, and the temporal space of *T* of the size $[t_b, t_e]$, where $t_b < t_e$.

[Definition 16] (GTS Block)

Any GTS can be defined as a GTS Block (GTSB).

[Definition 17] (Types of Blocks in GTS)

There are 4 types of blocks in GTS as follows:

1) *System block* (SB): This is the GTS block to represent a system in δ-Calculus. It consists of process blocks and their interactions in GTS.
2) *Process block* (PB): This is the GTS block to represent a process in a system. It consists of actions or interacts.
3) *Action block* (AB): This is the GTS block to represent an action in a process. An action can interact synchronously with another action in another process for communication and movements.
4) *Interaction block* (IB): This is the GTS block to represent a synchronous interaction between a process and another process.

[Definition 18] (Relations/Operations for Block)

The block relations an operations are defined as follows for $B_1 = \langle G_1, T_1 \rangle = \langle [g_{l,1}, g_{u,1}], [t_{l,1}, t_{u,1}] \rangle$ and $B_2 = \langle G_2, T_2 \rangle = \langle [g_{l,2}, g_{u,2}], [t_{l,2}, t_{u,2}] \rangle$:

1) Relations:

i) $=$(Equal): $B_1 = B_2$, if $G_1 = G_2 \wedge T_1 = T_2$.

ii) $<$(No-overlap): $B_1 < B_2$, if $G_1 < G_2 \wedge T_1 < T_2$.

iii) $\leq$(Geo-Contingent): $B_1 \leq B_2$, if $G_1 = G_2 \wedge T_1 \leq T_2$.

iv) $\sqsubset$(In-overlap): $B_1 \sqsubset B_2$, if $G_1 \sqsubset G_2 \wedge T_1 \sqsubset T_2$.

v) $\sqsubseteq$(Overlap): $B_1 \sqsubseteq B_2$, if $G_1 \sqsubset G_2 \wedge T_1 \sqsubseteq T_2$.

2) Operations:

i) $\sqcup$ (Union):

• $B_1 \sqcup B_2 = \emptyset$, if $B_1 < B_2 \vee B_1 \leq B_2$;
$= B_2$, if $B_1 \sqsubset B_2$;
$= B_3 = \langle G_2, T_3 \rangle$ if $B_1 \sqsubseteq B_2$,
where $T_3 = T_1 \sqcup T_2$.

ii) $\sqcap$ (Intersection):

• $B_1 \sqcap B_2 = \emptyset$, if $B_1 < B_2 \vee B_1 \leq B_2$;
$= B_1$, if $B_1 \sqsubset B_2$;
$= B_3 = \langle G_1, T_3 \rangle$ if $B_1 \sqsubseteq B_2$,
where $T_3 = T_1 \sqcap T_2$

iii) $-$ (Difference) :

i. $B_1 < B_2$ (No-overlap): $B_1 - B_2 = B_1$

ii. $B_1 \leq B_2$ (Contingent): $B_1 - B_2 = B_1$

iii. $B_1 \sqsubset B_2$ (In-overlap): No possible by the definition of GTS Block.

iv. $\sqsubseteq$ (Overlap): Not possible by the definition of GTS Block.

[Definition 19]] (GTS Sub-Block)

For $B_1 = \langle G_1, T_1 \rangle$ and $B_2 = \langle G_2, T_2 \rangle$, B_1 is a sub-block of B_2 if $B_1 \sqsubset B_2$.

[Definition 20] (Functions for Blocks)

1) General

• *id*(*B*), *name*(*B*), *time*(*B*), *geo*(*B*): Return the identifier, name, time interval and the geo interval of Block *B, respectively*.

2) System Block:

• *process*(*S*), *process*(*S,t*), *process*(*S,T*), *process*(*S,G*) : Return a set of processes in System Block *S*, at the time *t*, at the time interval *T*, and in the geo interval, respectively.

3) Process Block:

• *action*(*P*), *action*(*P,t*), *action*(*P,T*) : Return a set of actions in Process Block *P*, at the time t, and in the time interval, respectively.

• *parent*(*P*), *child*(*P*), *ancestor*(*P*), *descendent*(*P*), *system*(*P*): Return the parent, child, ancestor, descendent process(es), and the system process block of Process Block *P*.

4) Action Block:

i) *type*(*A*) : Returns the type of Action Block, i.e., α, β, γ.

ii) *process*(*A*), *system*(*A*): Return the owner process and the system block of Action Block *A*.

iii) Precedence:

• *next*(A), *previous*(A), *alternative*: Return the next, previous and the alternative action block of Action Block *A*.

5) Interaction Block:

i) *type*(*I*) : Returns the type of Interaction Block *A*, i.e., α, β, γ.

ii) *mode*(*I*) : Returns the mode of Interaction Block *A*, i.e., Synch, Asynch.

6) Movement Interaction Block:

i) *type*(I_δ) : Returns the type of Interaction Block *A*, i.e., Active, Passive.

ii) *mode*(*I*) : Returns the mode of Interaction Block *A*, i.e., Synch, Asynch.

[Definition 21] (Block Set Relations/Operations)

Block Set Relations and Operations are defined as follows for $\mathbb{S}_1 = \{B_{1,1}, \cdots, B_{1,m}\}, \mathbb{S}_2 = \{B_{2,1}, \cdots, B_{2,n}\}$:

1) Relations: $=, \neq, \subset, \subseteq$
2) Operations: $\cap, \cup, \ ^c, -$

[Notation 1] ('.' Infix Notation for Inclusion)

For simplicity, the following notations will be used to indicate each type of entities in δ-Calculus based on inclusion relations:

1) System: S_i
2) Process: $S_i.P_j$
3) Action: $S_i.P_j.A_k$
4) Interaction: $S_i.I_l = \langle P_{j,1}.A_{k,1}, P_{j,2}, A_{k,2} \rangle$

There are a number of propositions that are restricted by the syntax and semantic rules of δ-Calculus.

[Proposition 4] (Syntax Restrictions)

1) All $A_i, A_j \in P$, $A_i \sqcap A_j = \emptyset$.
2) If $P||Q$, $geo(P) < geo(Q) \wedge time(P) \sqsubseteq time(P)$.
3) If $P[Q]$, $Q \sqsubset P$.

(Proof):

1) Actions in *P* cannot be overlapped in any space in any time.
2) If *P* || *Q*, there is no overlap in space, but in time.
3) If *P*[*Q*], there is an overlap of *Q* over *P* in space during a period of time.

[Proposition 5] Semantic Restrictions

1) τ : Communication:
   i) If $I_\tau = \langle P.A_x, Q.A_v \rangle$ and $P \neq Q$, then $block(I_\tau) = \langle G, T \rangle =$
   $\langle [lower(block(A_x)), upper(block(A_y))], time(A_x) \sqcap time(A_y) \rangle$
   $\vee$
   $\langle [lower(block(A_y)), upper(block(A_x))], time(A_x) \sqcap time(A_y) \rangle$
2) δ : movements
   i) If $I_{active,in} = \langle P.in\ Q, Q.P\ in \rangle$ and $P \neq Q$, then $block(I_{active,in}) = \langle G, T \rangle =$
   $\langle [lower(block(in\ Q)), upper(block(P\ in))], time(in\ Q) \sqcap time(P\ in) \rangle \wedge$
   $block\left(process(block(in\ Q))\right)$
   $< block\left(process(block(Q\ in))\right)$
   $\wedge process\left(next(block(in\ Q))\right)$
   $\sqcap process\left(next(block(Q\ in))\right)$
   $= process\left(next(block(in\ Q))\right)$
   ii) If $I_{passive,in} = \langle P.Q\ get, Q.get\ P \rangle$ and $P \neq Q$, then $block(I_{passive,in}) = \langle G, T \rangle =$
   $\langle [lower(block(Q\ get)), upper(block(get\ P))], time(Q\ get) \sqcap time(get\ P) \rangle \wedge$
   $block\left(process(block(Q\ get))\right)$
   $< block\left(process(block(get\ P))\right)$
   $\wedge process\left(next(block(Q\ get))\right)$
   $\sqcap process\left(next(block(get\ P))\right)$
   $= process\left(next(block(Q\ get))\right)$
   iii) If $I_{active,out} = \langle P.out\ Q, Q.P\ out \rangle$ and $P \neq Q$, then $block(I_{active,out}) = \langle G, T \rangle =$
   $\langle [lower(block(out\ Q)), upper(block(P\ out))], time(out\ Q) \sqcap time(P\ out) \rangle \wedge$
   $block\left(process(block(out\ Q))\right)$
   $\sqsubset block\left(process(block(P\ out))\right)$
   $\wedge process\left(next(block(out\ Q))\right)$
   $< process\left(next(block(P\ out))\right)$
   iv) If $I_{passive,out} = \langle P.Q\ put, Q.put\ P \rangle$ and $P \neq Q$, then $block(I_{passive,out}) = \langle G, T \rangle =$
   $\langle [lower(block(Q\ put)), upper(block(put\ P))], time(Q\ put) \sqcap time(put\ P) \rangle \wedge$
   $block\left(process(block(Q\ put))\right)$
   $\sqsubset block\left(process(block(put\ P))\right) \wedge$
   $process\left(next(block(Q\ put))\right)$
   $< process\left(next(block(put\ P))\right)$
3) ρ : process control (Omitted Intentionally).

(Proof):

1) For communication (τ), *Q* and *P* must be in the same time period.
2) For movements (δ),
   - For active in movement, Q and P must be in the same space at the same time period.
   - For active out movement, Q must be in P or P must be in Q at the same time period.
   - For passive in movement, Q and P must be in the same space at the same time period.
   - For passive out movement, Q must be in P or P must be in Q at the same time period.
3) For control,
   - For terminate, suspend and wake, Q must be in P or P must be in Q at the same time period.

In δ-Calculus, a system *S* can be represented as a tuple that consists of a set of processes, a set of inclusion relations among processes and a global clock, where each process consists of a sequence of timed actions and a local clock.

[Example 4] GTS Diagram for PBC Example

A GTS diagram for PBC example is shown in Fig. 3. It is generated from simulating the left-most path from the execution model from Fig. 2, which is the case for the τ_1 communication.

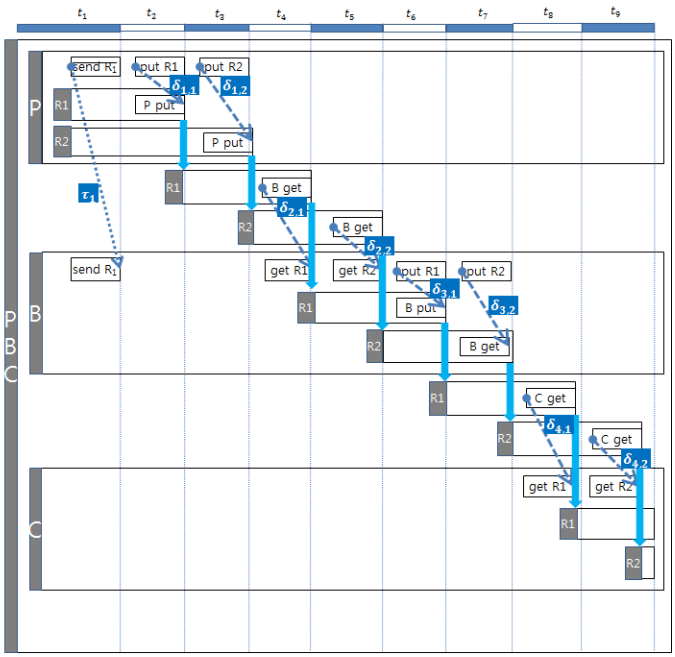

Fig. 3 GTS Diagram for PBC Example

B. GTS Logic

A) Syntax

GTS Logic is a first-order logic that deals with relationships among blocks in GTS.

[Definition 22] (Alphabet for GTS Logic)

1) Logical symbols:
   i) Quantitative symbols: $\exists, \forall$
   ii) Logical connectives: $\wedge, \vee, \rightarrow, \leftrightarrow, \neg$
   iii) Parenthesis, brackets, other punctuation symbols.
   iv) An infinite set of variables: x, y, z
   v) An equality symbol: $=$
2) Non-logical symbols
   i) Predicate symbols: P, Q, R (P_j^i)
   ii) Function symbols: f, g, h (f_j^i)
   iii) Constants: a, b, c
      i. Integer: 0, 1, 2, …
      ii. String: “a”, “aa”, …, “abc”, … .

[Definition 23] (Formulation Rules for GTS Logic)

1) Terms: Inductively defined by the following rules:
   i) Variables: Any variable is a term, such as, x, y, z.
   ii) Functions: Any expression $f(t_1, \dots, t_n)$ of n-arguments is a term and f is a function symbol of valence n is a term.
2) Formulas: Inductively defined by the following rules:
   i) Predicate symbols: It P is an n-ary predicate symbol and $t_1, \dots, t_n$ are terms, then $P(t_1, \dots, t_n)$ is a formula.
   ii) Equality: If t_1, t_2 are terms, then $t_1 = t_2$ is a term.
   iii) Negation: If φ is a formula, then ¬φ is a formula.
   iv) Binary connectives: If φ, ψ are formulas, then φ ⊡ ψ is a formula, where ⊡ can be ∧,∨,→, or ↔.
   v) Quantifier: If φ is a formula and x is a variable, then ∃xφ(x), ∀xφ(x) are formulas.

B) Predicates

[Definition 24] (Predicates for GTS Logic)

There are a number of predicates for GTS Logic as follows:

1) Membership: $\exists x. Member(X, x)$
2) Block Relation: $\exists x. Relation(a, x)$
   i) Process: $\exists x. Process(S, x)$
   ii) Action: $\exists x. Action(P, x)$
   iii) Interaction: $\exists x. Interaction(S, x)$; $\exists x. Interaction(P, x)$; $\exists x. Interaction(A, x)$
3) Precedency Relation: $\exists r. \text{Relation}(a, b, r)$
   i) Process: $\exists x. Relation(P, Q, x)$
   : parallel; choice; parent; child; anc; desc
   ii) Actions: $\exists x. Relation(a_i, a_j, x)$
   : next; previous; alternative; interaction
   iii) Interaction: $\exists x. Interaction(a_i, a_j, x)$
   : communication, movement, control
   iv) Interaction: $\exists x. Movement(a_i, a_j, x)$
   : active-in, passive-in, active-out, passive-Out.

C) Secure Property

[Definition 25] (Security Property)

Security is defined as the property that a system is free from unauthorized access.

[Definition 26] (Safety Property)

Safety is defined as the property that a system is free from unexpected behavior.

[Definition 27] (Secure System)

A system with security and safety properties is defined as a *secure* system.

D) Verifiability

[Proposition 6] Verification for Secure System

GTS Logic can be used to verify secure system.

(Proof):

All the secure requirements of a system can be represented with GTS Logic formulas, which can be proved to be true or false.

[Example 5] Interactions in PBC Example

There are two sets of interaction from the execution model of the PBC example in Fig. 2, as follows:

1) Communication (τ):
   i) $\tau_1 = \left(P.PB(\overline{Send\ R_1}), B.PB(Send\ R_1)\right)$ in T_1.
   ii) $\tau_2 = \left(P.PB(\overline{Send\ R_2}), B.PB(Send\ R_2)\right)$ in T_6.
2) Movements(δ):
   i) $\delta_{1,1} = \left(P.put\ R1, R1.P\ put\right)$ in T_2 and T_8.
   ii) $\delta_{1,2} = \left(P.put\ R2, R2.P\ put\right)$ in T_3 and T_7.
   iii) $\delta_{2,1} = \left(B.get\ R1, R1.B\ get\right)$ in T_4 and T_{10}.
   iv) $\delta_{2,2} = \left(B.get\ R2, R2.B\ get\right)$ in T_5 and T_9.
   v) $\delta_{3,1} = \left(B.put\ R1, R1.B\ put\right)$ in T_{11}.

vi) $\delta_{3,2} = (B.put\ R2, R2.B\ put)$ in T_{12}.
vii) $\delta_{4,1} = (C.get\ R1, R1.B\ get)$ in T_{13}.
viii) $\delta_{4,2} = (C.get\ R2, R2.B\ get)$ in T_{14}.

[Example 6] (Requirements for PBC Example)

There is a list of safety requirements for the PBC example as follows:

1) $Rq1 = \tau_1 \rightarrow \forall i \in I: (\delta_{i,1} < \delta_{i,2})$ $[I = \{1,2,3,4\}]$
   - After τ_1, $R1$ must move ahead of $R2$ all the time.
2) $Rq2 = \tau_2 \rightarrow (\forall j \in J(\delta_{j,2} < \delta_{j,2}) \wedge \forall k \in K: (\delta_{j,1} < \delta_{j,2}))$ $[J = \{1,2\}, K = \{3,4\}]$
   - After τ_2, $R2$ must move ahead of $R1$ out of P and in of B, and $R1$ must move ahead of $R2$ afterward, all the way to C.
3) $Rq3 = \tau_1 \vee \tau_2 \rightarrow \forall j \in J: (\delta_{i,1} < \delta_{i,2})$ $[J = \{3,4\}]$
   - After τ_1 or τ_2, once both $R1$ and $R2$ move to B, $R1$ must move ahead of $R2$ afterward, all the way to C.

Notice that $Rq3$ is derived from $Rq1$ and $Rq2$.

[Example 7] (Verification of Requirements)

All the requirements in Example 6 are satisfied for the first normal case of the τ_1 communication, shown in Example 5 with Fig. 3. It is clear that the requirements are satisfied for the second normal case of the τ_2 communication, shown in the right-most path of the execution model for the PBC example, shown in Fig. 2. The other two abnormal cases are not applicable. Therefore it can be concluded that the requirements for the example are satisfied for all the cases of the execution.

IV. Implementation

A. SAVE Tool

A tool, called SAVE (*Specification, Analysis, Verification Environment*), for δ-calculus has been developed on ADOxx [5], as shown in Fig. 2. It consists of four basic components as follows:

1) Modeler: It provides capability to specify System and Process Views.
2) EM Generator: It generates an execution model, in *Execution Tree* (ET), for the views and makes each path of the model to be selected for simulation.
3) Simulator: It generates a model for the selected simulation, in a GTS diagram.
4) Verifier: It verifies the secure requirements of the system by model-checking on the diagrams.

The graphical representations of the models in SAVE are designed by the ADOxx Development Tool, and the procedures of its components are built from the ADOxx libraries. The detailed logics of the procedures are programmed in the ADOScript language. ADOxx provides three layers to implement mechanisms and algorithms for SAVE:

1) First layer: The pre-defined functionality, a basic set of features most commonly used by modeling tools.
2) Second Layer: Approximately 400 APIs for the generation of objects, editing of their properties, etc.
3) Third layer: Ways of interaction to outside of ADOxx. The simple interaction is by exporting and importing XML files.

SAVE uses the functionalities of the first layer to implement the graphical elements and attributes of the graphic models, and it uses those of the second layer to implement Modeler, EM Generator, Simulator and Verifier.

B. Example

The PBC example from the previous sections are demonstrated in SAVE. The source code for the example is shown in Fig. 1.

1) System View

The system view for the PBC example is shown in Fig. 4. As stated, there are five processes running, namely, *P*, *B*, *C*, *R*1 and *R*2, in parallel. Notice that *R*1 and *R*2 are running in *P*.

2) Process View

The process views for the PBC example are shown in Fig. 5. As stated, each process view specifies visually the detailed operational requirements of the individual process in the PBC example.

3) Execution View

The execution view of the PBC example is shown in Fig. 6. It shows four possible execution cases: two for normal termination and two for abnormal termination. The cases for normal termination are the ones for safe communication based on synchronization between *P* and *B*: *Send R*1 and *Send R*2. The cases for abnormal termination are the ones for unsafe communication, that is, deadlock, based on unexpected synchronization between *P* and *B*: *Send R*1 for *Send R*2, and *Send R*2 for *Send R*1.

4) Simulation View

In order to generate the simulation view for each execution case, the execution path can be selected from the dialog box from the execution view, as shown in Fig. 6. Once the path is selected, the processes from the process view are simulated under the condition for the selected path. Fig. 7 shows the simulation view for the first normal case shown as the right-most path in the execution view.

5) Verification View

The verification view for the simulation view in Fig. 7 is shown in Fig. 8. This is a result of verification for the validity of safety requirements for the PBC example. As stated, there are a number of dependencies and restrictions that the PBC example must follow. For example, the movement of R1 must be

performed after the communication between *P* and *B*, and the movement of *R*1 must be ahead of the movement of *R*2.

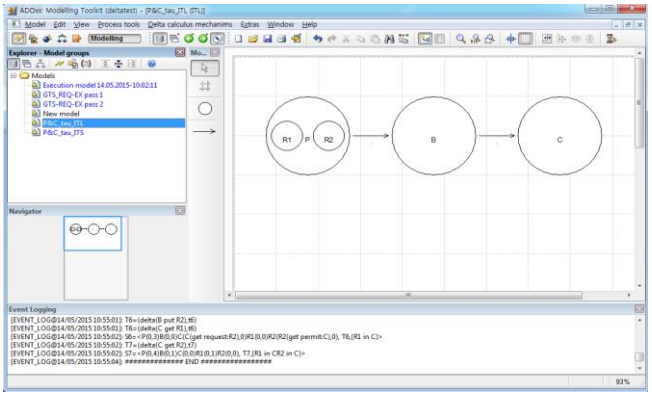

Fig. 4 System View for PBC Example in SAVE

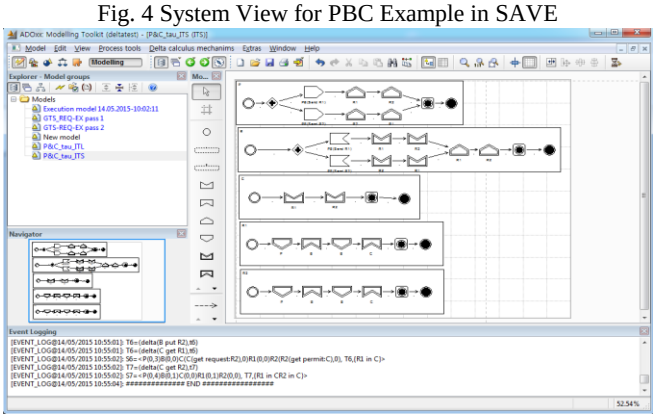

Fig. 5 Process View for PBC Example in SAVE

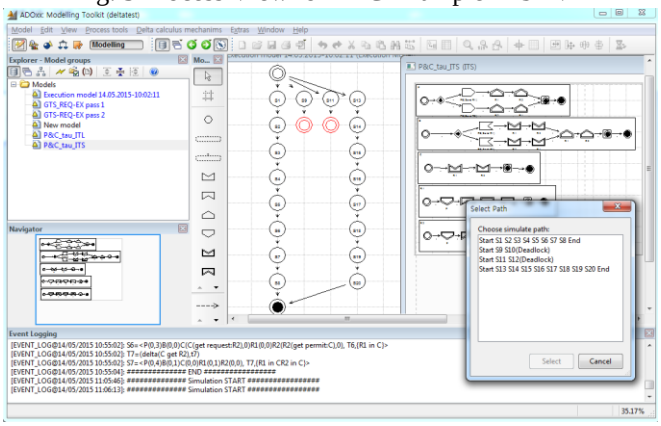

Fig. 6 Execution View for PBC Example

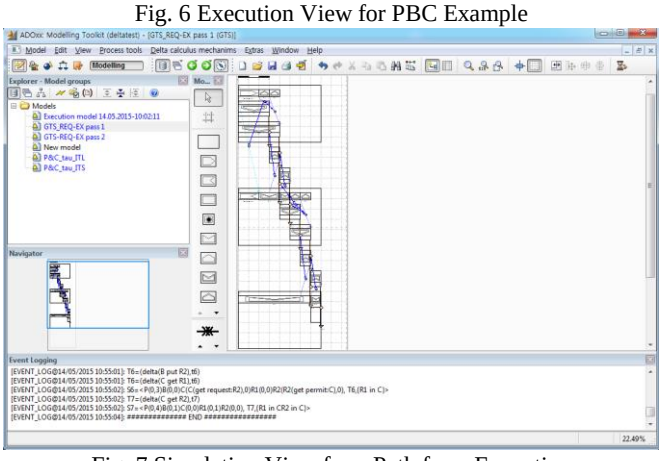

Fig. 7 Simulation View for a Path from Execution

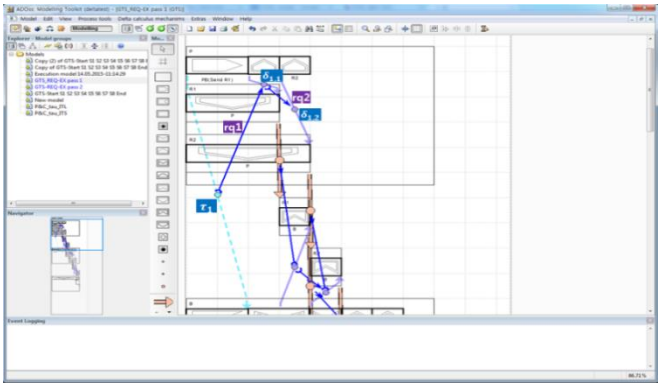

Fig. 8 Verification View for Simulation View

In the figure, the resources *R*1 and *R*2 are passed to *B* by *P* in order after communicating with *B* to determine which resource to be passed first. If *R*1 is determined to be passed first (τ_1), *R*1 must be passed first to *B* by *P* ($\delta_{1,1}$ for *R*1), followed by *R*2 being passed to *B* by *P* next ($\delta_{1,2}$ for *R*2). And *R*1 and *R*2 must be passed to *C* by *B*, in that order, that is, $\delta_{3,1}$ (for *R*1) is followed by $\delta_{3,2}$ (for *R*2). This is one of the requirements, that is, $\tau_1 \rightarrow \delta_{1,2} < \delta_{1,2} < \delta_{3,1} < \delta_{3,2}$ in GTS Logic, which is visually specified in Fig. 8 with the icons for the $<$ and $\rightarrow$ dependency relations. If the requirements are satisfied, they are colored in blue. If not, in red. In the figure, the requirement is known to be satisfied, since its color is determined to be blue after verifying all the requirements to be true.

V. Conclusion

This paper presented a dual method to specify and verify IoT systems. There were three basic phases of the approach: 1) specifications of both dynamic properties, such as, operational requirements, in δ-Calculus and the static properties, such as, safety requirements, in GTS Logic, 2) and the execution model and simulations, and 3) verifications. The method was realized in a visual environment, namely SAVE, and its efficiency and effectiveness were demonstrated with the PBC example on SAVE. Five views were provided in SAVE to show the consistency of the methods for visualization: System, Process, Execution, Simulation and Verification Views. It can be considered to be one of the most innovative dual methods and tools to model IoT systems in terms of specification and verification.

Future research will include the application of the real industrial examples to demonstrate the feasibility of the method on SAVE. For example, SAVE must have capability to handle modularization and scalability of extensive size of the execution models, similar to the EMS example showed in Fig. 14 [6]. The example handles more than 17,000 execution paths in the model.

Acknowledgment

This work was supported by Basic Science Research Programs

through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2010-0023787), and the MISP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2016-H8501161012000100 2) supervised by the IITP (Institute for Information & communications Technology Promotion), and Space Core Technology Development Program through the NRF (National Research Foundation of Korea) funded by the Ministry of Science, ICT and Future Planning (NRF-2014M1A3A3A02034792), and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A3A01019282).

REFERENCES

[1] Rajeev Alur, Emery Berger, Ann W.Drobnis, Limor Fix, Kevin Fu, Gregory D.Hager, Daniel Lopresti, Klara Nahrstedt, Elizabeth Mynnat, Shwetak Patel, Jennifer Rexford, Jhon A.Stankovic, and Benjamin Zorn, Systems Computing Challenges in the Internet of Things, A white paper prepared for the Computing Community Consortium, 2015.

[2] A. Whitmore , A. Agarwal and Li Da Xu, The Internet of Things – A survey of topics and trends, *Information Systems Frontiers*, vol 17, issue 2, pp 261-274, Springer, April 2015.

[3] Y. Choe and M. Lee, δ -Calculus: Process Algebra to Model Secure Movements of Distributed Mobile Processes in Real-Time Business Application, 23rd European Conference on Information Systems, 2015.

[4] Y. Choe, W. Choi, G. Jeon and M. Lee, A Tool for Visual Specification and Verification for Secure Process Movements, eChallenges e-2015, November 2015.

[5] H. Fill and D. Karagiannis, On the Conceptualisation of Modeling Methods Using the ADOxx Meta Modeling Platform, Enterprise Modeling and Information Systems Architectures 8(1), pp.4-25, 2013.

[6] W. Choi, Y. Choe and M. Lee, A Reduction Method for Process and System Complexity with Conjunctive and Complement Choices in a Process Algebra, IEEE 39th Annual International Computer, Software & Applications Conference, 2015.

Sunghyeon Lee

Received Bachelor Degree of Engineering in Industrial and Information Systems Engineering from Chonbuk National University, Republic of Korea, February 2015. Currently working for Master Degree of Division of Electronics and Information Engineering at Chonbuk National University, Korea.

Yeongbok Choe

Received Bachelor and Master Degrees of Computer Science and Engineering from Chonbuk National University, Republic of Korea, in 2013 and 2015, respectively. Currently working for PhD Degree of Computer Science and Engineering at Chonbuk National University.

Moonkun Lee

Received Bachelor Degree of Computer Science from Pennsylvania State University, USA, in 1989, Master and PhD Degrees from The University of Pennsylvania, USA, in 1992 and 1995, respectively. Developed SRE (SW Re/reverse-engineering Environemnt) at CCCC, USA, between 1992 and 1996 as Computer Scientist. Currently Professor at Chonbuk National University, Republic of Korea, since 1996. Research areas include formal methods, SW re/reverse-engineering, real-time systems, operating systems, formal languages, parallel equational languages, compilers, etc.