

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800

Open access books available

122,000

International authors and editors

135M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



dT-Calculus: A Formal Method to Specify Distributed Mobile Real-Time IoT Systems

Sunghyeon Lee, Yeongbok Choe and Moonkun Lee

Additional information is available at the end of the chapter

<http://dx.doi.org/10.5772/intechopen.75138>

Abstract

In general, process algebra can be the most suitable formal method to specify IoT systems due to the equivalent notion of processes as things. However there are some limitations for distributed mobile real-time IoT systems. For example, *Timed pi-Calculus* has capability of specifying time property, but is lack of direct specifying both execution time of action and mobility of process at the same time. And *d-Calculus* has capability of specifying mobility of process itself, but is lack of specifying various time properties of both action and process, such as, *ready time*, *timeout*, *execution time*, *deadline*, as well as priority and repetition. In order to overcome the limitations, this paper presents a process algebra, called, *dT-Calculus*, extended from *d-Calculus*, by providing with capability of specifying the set of time properties, as well as priority and repetition. Further the method is implemented as a tool, called *SAVE*, on ADOxx meta-modeling platform. It can be considered one of the most practical and innovative approaches to specify distributed mobile real-time IoT systems.

Keywords: dT-Calculus, process algebra, mobility, time, SAVE, ADOxx

1. Introduction

The main characteristics of distributed mobile real-time IoT systems can be movement of things on some geographical space and real-time communication among them with deadlines [1]. Therefore it is necessary to specify these characteristics with formal methods during design phase of the system development process, and process algebra is known to be best suitable for the specification of the systems since things can be considered as processes and the characteristics can be depicted as both the movements of processes and the timed communications

among them [2]. For example, the most suitable process algebras for IoT systems can be as follows:

1. *Timed pi-Calculus* [3]: It is the timed version of the existing *pi-Calculus* [4], which expresses process movements indirectly by using the notion of *value passing*. It allows *time-stamp* and *clock* to be passed additionally during value passing, with which the temporal requirements of the process movements can be specified.
2. *Timed Mobile Ambient* [5]: It is the timed version of the existing *Mobile Ambient* [6], where process can move by ambient with *in*, *out*, and *open* capabilities. In contrast to *pi-Calculus*, it is based on the semantics of autonomous movement, and makes timed specification possible by adding time property to the movement.
3. *d-Calculus* [7]: This is a process algebra that can express direct process movements into or out of other processes by using the four types of synchronous movements with simple temporal conditions: a bound of the minimum and maximum limits. It naturally allows process nesting by the resulting inclusion relations among processes.

However it is noticed that there are fundamental limitations in the above process algebra to specify the main characteristics of distributed mobile real-time IoT systems due to lack of both full description power of mobile and temporal properties, as follows:

1. *Timed pi-Calculus*: It allows various types of temporal requirements to be specified, but it is not possible to specify directly both the actual execution time of action itself and the type of its movement in the same requirements.
2. *Timed Mobile Ambient*: It is possible to specify temporal requirements by adding temporal property to ambient, but it is difficult to understand intuitively process synchronization since the synchronization is represented by the movement of the ambient.
3. *d-Calculus*: It allows various types of temporal requirements to be specified, but only simple types of temporal requirements for process movements are possible. For example, a temporal bound of the minimum and maximum limits. It results in limited specification of the temporal requirements of the movements as well as analysis of the requirements.

In order to overcome the limitations, this paper proposes process algebra, namely, *dT-Calculus*, which is the timed version of *d-Calculus*, extended for more specific temporal specification and analysis of the requirements of the IoT systems. More specifically, *dT-Calculus* allows the temporal properties of the actions of processes to be expressed as follows:

- *Ready time*: The time needed before execution of an action or a process.
- *Timeout*: The maximum waiting time up to the actual execution of an action or a process, after the execution will be ready with *ready time*.
- *Execution Time*: The actual execution time of an action or a process.
- *Deadline*: The time that the execution of action is to be terminated.
- *Period*: Period for repetition of an action or a process.

These specific temporal properties allow various types of temporal requirements of process movements and communications over the IoT environment to be specified and analyzed, without modifying any types of the process movements and communications from d-Calculus.

This paper is organized as follows. Section II introduces some of the existing process algebras with temporal properties. Section III introduces the basic algebra for dT-Calculus, that is, d-Calculus. Section VI describes syntax and semantics of dT-Calculus, focusing on its temporal properties. Section V demonstrates usability of dT-Calculus with a simple IoT example. Section VI shows some comparison of dT-Calculus with other process algebras. Section VII introduces a tool, called SAVE [8, 9], which is developed on ADOxx meta-modeling platform, to specify and analyze the temporal requirements of the process movements with dT-Calculus. Finally conclusions will be made and some of future researches will be discussed.

2. Related research

2.1. Timed pi-Calculus

One of the best known process algebra to specify the temporal properties is Timed pi-Calculus. It is the timed version of pi-Calculus, adding the temporal properties to process movements. **Figure 1** shows the syntax of Timed pi-Calculus.

In the *send* and *receive* actions of the calculus, t_c and c represent *time-stamp* and *clock* used for creating of the time-stamp, respectively. Further δ and γ represent *temporal restriction condition* and *clock reset*, respectively. The process specification with temporal restriction condition is to be used as follows:

$$P = (c < 2)\bar{x}\langle y, t_c, c \rangle.P' \quad (1)$$

It implies that, in 2 time units after *clock* c is reset, *name* y can be transmitted through *channel* x in t_c .

The notion of clock in Timed pi-Calculus is based on local clock concept, which allows various kinds of temporal restriction conditions. For example,

$P ::= M$	message	$M ::= \delta\gamma\bar{x}\langle y, t_c, c \rangle.P$	send
$(P \mid P')$	composition	$\delta\gamma x(\langle y, t_c, c \rangle).P$	receive
$!P$	replication	$\delta\gamma\tau.P$	internal action
$(z)P$	restriction	0	inactive process
$[x = y]P$	match(name)	$M + M'$	choice
$[c = d]P$	match(clock)		

Figure 1. Syntax of Timed pi-Calculus.

$$Q = (e > 5)(d - t_z \leq 3)x(\langle z, t_z, d \rangle).Q' \quad (2)$$

It specifies two temporal conditions with clock: $(e > 5)$ represents a condition for a local clock e , and $(d - t_z \leq 3)$ represents a temporal condition related to a receiving message. d and t_z are the temporal conditions on the clock for the receiving message and its time-stamp, but, since the clock ticks continuously, $(d - t_z \leq 3)$ implies the temporal condition that the message should be transmitted in 3 time units.

The mobile property of Timed pi-Calculus is represented indirectly by changing the state of channel connection among processes through passing the connecting channel names. For example,

$$\bar{y}x.P' | y(z).Q' | R \xrightarrow{\tau} P' | Q' \{x/z\} | R \quad (3)$$

As shown in **Figure 2**, it represents the state of P and R , connected by x , to be changed to the state of Q and R , newly connected by x , after passing the name x to Q by P through the channel y . Obviously the connection between of P and R is invalid since there is no x in P .

2.2. Timed Mobile Ambient

Timed Mobile Ambient is another process algebra to specify process movements and temporal properties. It is the timed version of Mobile Ambient. **Figure 3** shows the syntax of Timed Mobile Ambient.

In Timed Mobile Ambient, 0 represents the process with no action. n in $n^{\Delta t}[P]^{\mu}$ implies the location where Process P executes, and Δt does that P should terminate its execution in t .

If $t > 0$, then ambient $n^{\Delta t}[P]^{\mu}$ is equal to $n[P]$. If a timer becomes 0 by $t = 0$, then $n^{\Delta t}[P]^{\mu}$ can be represented as a pair of $(n^{\Delta t}[P]^{\mu}, Q)$, where Q is a safe process, implying that, in case that $n^{\Delta t}[P]^{\mu}$ is not completed in time or timed out, a safe process Q can be activated in order to handler the time-out case of $n^{\Delta t}[P]^{\mu}$. For example, if the *open* n capability does not occur in the time t , ambient $n^{\Delta t}[P]^{\mu}$ is deactivated, and a safe process Q is activated instead as a handler. If $Q = 0$, then $n^{\Delta t}[P]^{\mu}$ can be simple enough to represent $(n^{\Delta t}[P]^{\mu}, Q)$.

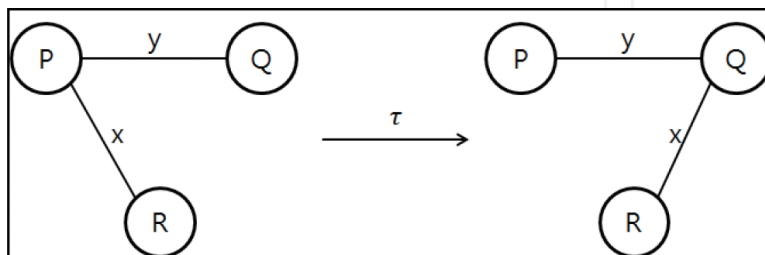


Figure 2. Movement in Timed pi-Calculus.

$a, p,$	ambient tags	$P, Q ::=$	processes
c	channel name	$ 0$	inactivity
n, m	ambient names	$ M^{\Delta t}. (P, Q)$	movement
x	variable	$ (n^{\Delta t} [P]^{\mu}, Q)$	ambient
		$ P Q$	composition
$M ::=$	capabilities	$ (vn: Amb[\Gamma])P$	restriction
$ in\ n$	can enter n	$ c^{\Delta t!} < m: Amb[\Gamma] >. (P, Q)$	output action
$ out\ n$	can exit n	$ c^{\Delta t?} (x: Amb[\Gamma]). (P, Q)$	input action
$ open\ n$	can open n	$ * P$	replication

Figure 3. Syntax of Timed Mobile Ambient.

(R-GTProcess)	$\frac{P \rightarrow}{P \rightarrow \phi \Delta (P)}$	(R-Res)	$\frac{P \rightarrow Q}{(vn: Amb[\Gamma])P \rightarrow (vn: Amb[\Gamma])Q}$
(R-In)	$\frac{n^{\Delta t'} [in^{\Delta t} m. (P, P') Q]^{\alpha}, S' (m^{\Delta t''} [R]^{\mu}, S'') \rightarrow}{(m^{\Delta t'} [(n^{\Delta t'} [P Q]^{\mu}, S') R]^{\mu}, S'') \rightarrow}$	(R-Amb)	$\frac{P \rightarrow Q}{(n^{\Delta t} [P]^{\mu}, R) \rightarrow (n^{\Delta t} [Q]^{\mu}, R)}$
(R-Out)	$\frac{(m^{\Delta t'} [(n^{\Delta t''} [out^{\Delta t} m. (P, P') Q]^{\alpha}, S'') R]^{\mu}, S') \rightarrow}{(n^{\Delta t'} [P Q]^{\mu}, S'') (m^{\Delta t'} [R]^{\mu}, S') \rightarrow}$	(R-Par1)	$\frac{P \rightarrow Q}{R P \rightarrow R Q}$
(R-Com)	$\frac{c^{\Delta t!} < m: Amb[\Gamma] >. (P, Q) c^{\Delta t?} (x: Amb[\Gamma]). (P', Q') \rightarrow}{P P' \{m/x\}}$	(R-Par2)	$\frac{P \rightarrow P', Q \rightarrow Q'}{P Q \rightarrow P' Q'}$
(R-Open)	$\frac{n: Amb[\Gamma'], m: Amb[\Gamma], \Gamma <: \Gamma'}{(m^{\Delta t'} [open^{\Delta t} n. (P, P') (n^{\Delta t'} [Q]^{\mu}, S'') S'] \rightarrow (m^{\Delta t'} [P Q]^{\mu}, S'))}$	(R-Struct)	$\frac{P' \equiv P, P \rightarrow Q, Q \equiv Q'}{P' \rightarrow Q'}$

Figure 4. Reduction rules of Timed Mobile Ambient.

Tags are related to reductions, which are similar to execution rules, and are classified into active and passive ones. And μ is a neutral tag to represent whether a tag is active or passive. An active tag performs a reduction in a time unit by consuming capability, and a passive tag performs a series of reductions in time units. The reduction rule is defined in **Figure 4**.

The movement $M^{\Delta t}. (P, Q)$ is provided by the capability M , and followed by the execution of Process P . If the time becomes 0 as in $t = 0$, the safe process Q is executed instead of P .

An output action implies that Process P releases a name m on Channel c . An input action implies that that Process P brings a name from Channel c and binds it to a name n within the scope of P . Restriction does that a new unique name n is declared within the scope of P .

Since the communication method used in Timed Mobile Ambient is not direct, it is possible to define appropriate types for receivers in communication. The $Amb[\Gamma]$ in the restriction and the output and input actions is used to define such types.

Figure 5 shows a part of the Cab Protocol in Timed Mobile Ambient [5]. The basic scenario of the protocol is that *cab* takes on a *client* sending the signal *call* from the place *from*. If the call

$$\begin{aligned}
load\ client &= loading^{\Delta t_1} [out^{\Delta t_2} cab.in^{\Delta t_3} client]^{\mu} \\
call &= call^{\Delta t_7} [out^{\Delta t_8} client.out^{\Delta t_9} from.in^{\Delta t_{10}} cab.in^{\Delta t_{11}} from.load\ client]^{\mu} \\
recall &= recall^{\Delta t_{12}} [out^{\Delta t_{13}} cab.in^{\Delta t_{14}} from.in^{\Delta t_{15}} client]^{\mu} \\
call\ from\ client &= (call, recall)
\end{aligned}$$

Figure 5. Timed Mobile Ambient example.

from the client is not replied, the client should *recall*. The cab can be absent or full of customers, the client can be waiting for a cab at the specific place while sending signals or be on a cab. In order to specify the scenario, four processes are defined: *load client*, *call*, *recall*, and *call from client*.

In specification, Ambient *client* must enter *cab*, and *cab* can release Ambient *load client*. After Ambient *client* gets off *cab*, Ambient *from* looks for *cab* for another *client*'s transportation. If Ambient *from* finds *cab*, *client* gets on *cab* by the *R-In* reduction.

$$\left(call^{\Delta t_7} [in^{\Delta t_{10}} cab.in^{\Delta t_{11}} from. \dots]^a, recall \right) | cab^{\infty} []^{\mu} \rightarrow cab^{\infty} [call^{\Delta t_7} [in^{\Delta t_{11}} from. \dots]^p, recall]^{\mu} \quad (4)$$

If the timer Δt_7 of Ambient *call* is terminated before getting-on *cab*, Ambient *call* is released automatically. This kind of specification allows for Ambient *cab* and Ambient *call* not to contact each other in Δt_7 . After releasing Ambient *call*, a safe process can be executed by the *R-GTP* reduction.

$$\left(call^{\Delta t_7} [in^{\Delta t_{10}} cab.in^{\Delta t_{11}} from. \dots]^a, recall \right) \rightarrow recall \quad (5)$$

Once Ambient *recall* enters Ambient *client*, other *calls* will be informed for execution. The *recall* process will repeat itself until *load client* is released.

3. Preliminary research

d-Calculus is the process algebra developed to specify and analyze the process movements directly on geographical space. There are four types of movements in d-Calculus, all of which are synchronously defined.

3.1. Syntax

The syntax of d-Calculus is shown in **Figure 6** and is defined as follows:

1. Action: Actions performed by a process.
2. Priority: The priority of the process P represented by a natural number $n \geq 0$. The higher number represents the higher priority. Exceptionally, 0 represents the highest priority.

$P ::= A$	action	$M ::= m_t^p(k) P$	request
$P_{(n)}$	priority	$P m(k)_t$	permission
$P[Q]$	nesting		
$P\langle r_t \rangle$	channel	$m ::= in out get put$	movement types
$P + Q$	choice		
$P \parallel Q$	parallel	$C ::= new P$	create process
$P \setminus_t F$	exception	$kill P$	kill process
$A \cdot P$	sequence	$exit$	exit process
$A ::= \emptyset$	empty		
$r_t(msg)$	send		
$r_t(msg)$	receive		
M	movement		
C	control		

Figure 6. Syntax of d-Calculus.

3. Nesting: P contains Q . The internal process is controlled by its external process. If the internal process has a higher priority than that of its external, it can move out of the external without the permission of the external.
4. Channel: A channel r of P to communicate with other processes. t implies the time needed for the communication through the channel.
5. Choice: Only one of P and Q will be selected non-deterministically for execution.
6. Parallel: Both P and Q are running concurrently.
7. Exception: Execution of P , but F in case of violation of the deadline t .
8. Sequence: P follows after action A .
9. Empty: No action.
10. Send/Receive: Communication between processes, exchanging a message by a channel r . t represents deadline of the communication.
11. Request: Requests for movement. t , p and k represent deadline, priority and key, respectively.
12. Permission: Permissions for movement. t represents deadline.
13. Create process: Creation of a new internal process. The new process cannot have a higher priority than its creator.
14. Kill process: Termination of other processes. The terminator should have the higher priority than that of the terminate.
15. Exit process: Termination of its own process. All internal processes will be terminated at the same time.

Generally all the movements are synchronous. In order for a process to move in or out of another process, the moving process (*mover*) needs permission from the target process.

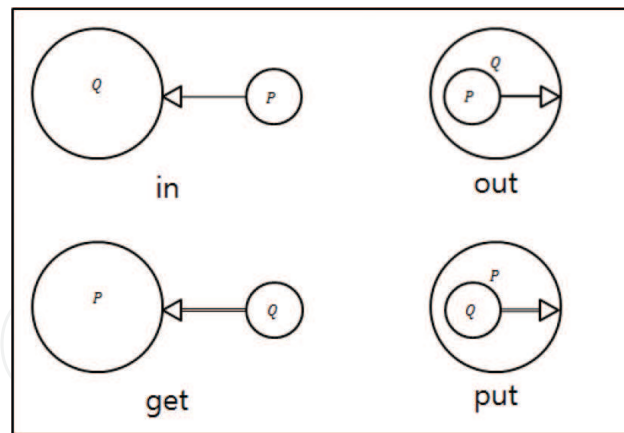


Figure 7. Pictorial view of d-Calculus movements.

Reversely, in order for a process to be moved in or out of another process forcefully, the moving process needs permission from the being-moved process (*movee*).

By means of the strict method of synchrony, the movements of processes can be controlled, and further the security and safety of the IoT systems can be guaranteed by pre-cautiously preventing insecure or unsafe movements.

3.2. Mobility

As stated, the process movement in d-Calculus occurs synchronously between the requesting process and the permitting process. It implies that the movement cannot be allowed without permission. It prevents any unplanned movement from occurring unexpectedly, and clarifies control of the movement explicitly. There are four types of such movements in d-Calculus as follows:

- *in*: A process moves into another process directly.
- *out*: A process moves out of another process directly.
- *get*: A process makes another process move into itself.
- *put*: A process makes another process move out of itself.

The types of movements can be pictorial depicted as shown in **Figure 7**.

4. dT-Calculus

dT-Calculus is the process algebra developed to specify and analyze the movements of things in the IoT systems with temporal restrictions directly on geographical space. In order to represent precise temporal properties explicitly, it extended the basic temporal property of the movements in d-Calculus to specify the different types of temporal properties for period and sporadic actions or processes, with the additional syntax and semantics accordingly.

4.1. Temporal properties

As shown in **Figure 8**, there are five temporal properties in dT-Calculus: *ready time*, *timeout*, *execution time*, *deadline*, and *period*. The first four properties are used to specify the temporal properties of sporadic actions or processes, and the last one is used to specify the temporal properties of periodic actions and processes inclusively. The definition of each property is as follows:

1. *Ready time*: It represents the waiting time for an action. At the point of the action in a process, the process was to wait in *ready time* before executing the action. No other or synchronous actions are possible during *ready time*.
2. *Timeout*: It represents the maximum waiting time for the actual execution of an action to be started after the action is ready for execution. If the waiting time in *ready time* is over and the partner for its synchronous action is not ready, the action cannot be executed. If the partner is ready for the action in *timeout*, the action can be executed. If not, the action will be in the state of *timeout*, the process will be in some fault state unless some proper handling action is not specified.
3. *Execution Time*: The time needed to execute an action. In case that the action can be performed in *timeout* after *ready time*, the action will be executed in *execution time* and be terminated. And then the next action will be available.
4. *Deadline*: The termination time for the execution of an action. All actions must be terminated in *deadline*. *Deadline* starts as *ready time* does. If the action is terminated in *deadline*, the process will be in some fault state. In order to prevent the process from being in the fault state, an exceptional handling must be specified accordingly.
5. *Period*: The duration of period for the execution of an action or process in repetition. The action will repeat itself after period of executing the action or process. This is an additional temporal property to specify the periodic action or process, different from the previous four temporal properties. The periodic action or process can be put into some fault state due to failure or *timeout* and *deadline*.

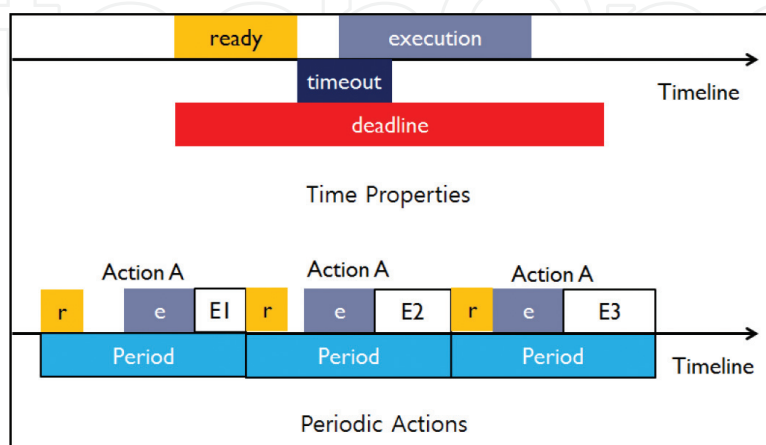


Figure 8. Time properties of dT-Calculus.

All actions and processes are defined or specified with these temporal properties. However the properties cannot be applied to some actions and processes. For example, *empty* action, no-time action, timed process, etc.

4.2. Syntax

The syntax of dT-Calculus is shown in **Figure 9**, and the extended notions from d-Calculus for temporality are as follows:

1. Timed action: The execution of an action with temporal restrictions. The temporal properties of $[r, to, e, d]$ represent *ready time*, *timeout*, *execution time*, and *deadline*, respectively. p and n are properties for periodic action or processes: p for period and n for the number of repetition.
2. Timed process: Process with temporal properties.
3. Exception: P will be executed. But F will be executed in case that P is out of timeout or deadline.

The biggest difference of dT-Calculus with d-Calculus is the notion of *timed* action and processes. In d-Calculus, the temporal property is simple, defined with a time interval in action or process: the boundary of the lower and upper time limits. However, in dT-Calculus, the property is divided into more specific properties, as described. In addition, the exceptions caused by the violation of the temporal properties are more specifically divided into the one by *deadline* and the one by *timeout*.

Consequently the separate notions for temporal properties for action and process in d-Calculus can be represented in one single notion and form of the properties in dT-Calculus.

If there is no temporal properties to be specified in an action, it will be considered to be $[0,-,1,-]$ by default. That is, there is no waiting time so that the action can be executed immediately, and infinite waiting for the synchronous co-action is possible without *timeout* and *deadline*.

$P ::= A$	action	$A ::= \emptyset$	empty
$A_{[r,to,e,d]}^{p,n}$	timed action	$r(\overline{msg})$	send
$P_{[r,to,e,d]}^{p,n}$	timed process	$r(msg)$	receive
$P_{(n)}$	priority	M	movement
$P[Q]$	nesting	C	control
$P\langle r \rangle$	channel	$M ::= m^p(k) P$	request
$P + Q$	choice	$P m(k)$	permission
$P \parallel Q$	parallel	$m ::= in \mid out \mid get \mid put$	movement types
$P \setminus F$	exception	$C ::= new P$	create process
$A \cdot P$	sequence	$kill P$	kill process
		$exit$	exit process

Figure 9. Syntax of dT-Calculus.

4.3. Semantics

The semantics of dT-Calculus for the temporal properties in action and process are defined as transition rules as shown in **Table 1**.

Each rule in the table is defined as follows:

1. *Tick-Time R*: The rule for *ready time* of an action. As time passes in *ready time*, the values of r and d decrease accordingly.
2. *Tick-Time TO*: The rule for *timeout* of an action. The action, not executing, but in waiting, decreases its *timeout* time accordingly as time passes.
3. *Tick-Time End*: The rule for termination of an action. After the execution of the action started and the value of e becomes 0, the next action can start.
4. *Tick-Time SyncE*: The rule for execution of an action. When an action and its partner co-action are executed synchronously, the values of e and d decrease accordingly as time passes.
5. *Tick-Time AsyncE*: The rule for *execution time* of an asynchronous action. In case of asynchronous action, there is no need for timeout: it goes into its own execution immediately just after *ready time* and the values of e and d decrease accordingly as time passes.

Tick-Time R	$\frac{-}{A_{[r,to,e,d]} \xrightarrow{\triangleright_1} A_{[r-1,to,e,d-1]}} (r \geq 1)$
Tick-Time TO	$\frac{-}{A_{[0,to,e,d]} \xrightarrow{\triangleright_1} A_{[0,to-1,e,d-1]}} (to \geq 1)$
Tick-Time End	$\frac{-}{A_{[0,to,0,d]} \cdot A' \xrightarrow{\triangleright_1} A'}$
Tick-Time SyncE	$\frac{A A' \xrightarrow{(\tau \vee \delta) \wedge \triangleright_1} A'' A'''}{A_{[0,to_1,e_1,d_1]} A'_{[0,to_2,e_2,d_2]} \xrightarrow{(\tau \vee \delta) \wedge \triangleright_1} A_{[0,to_1,e_1-1,d_1-1]} A'_{[0,to_2,e_2-1,d_2-1]}} (e_1 \geq 1 \wedge e_2 \geq 1)$
Tick-Time AsyncE	$\frac{-}{A_{[0,to,e,d]} \xrightarrow{\triangleright_1} A_{[0,to,e-1,d-1]}}$
Tick-Time P	$\frac{-}{P_{[r,to,e,d]} \xrightarrow{\triangleright_1} P_{[r,to,e,d-1]}}$
Timeout	$\frac{-}{A_{[0,0,e,d]} \setminus P \xrightarrow{\triangleright_1} P}$
Deadline	$\frac{-}{A_{[r,to,e,0]} \setminus P \xrightarrow{\triangleright_1} P}$
Period	$\frac{-}{A_{[r,to,e,d]}^{p,n} \xrightarrow{\triangleright_p} A_{[r,to,e,d]}^{p,n-1}} (n > 1)$
Period End	$\frac{-}{A_{[r,to,e,d]}^{p,1} \cdot A' \xrightarrow{\triangleright_p} A'}$

Table 1. Temporal semantics of dT-Calculus.

6. *Tick-Time P*: The rule for passage of time in process. Since the temporal property for a process uses only deadline in its temporal requirements, the value of e decreases accordingly as time passes.
7. *Timeout*: The rule for *timeout* to occur. When the value of to becomes 0, its timeout error will occur. However, when an exception for the *timeout* defines, its exception handling will be activated accordingly.
8. *Deadline*: The rule for violation of *deadline*. When the value of d becomes 0, its deadline error will occur. However, when an exception for the *deadline* defines, its exception handling will be activated accordingly.
9. *Period*: The rule for execution of a periodic action. The action will be executed again after the period passes, and the value of n will be decremented by 1.
10. *Period End*: The rule for termination of a periodic action. In case that the value of n is 1, no action will be repeated after the period passed over.

4.4. Laws

The laws for the additional temporal properties in dT-Calculus are shown in **Table 2**. The laws represent the notion and restrictions of temporal properties in dT-Calculus as follows:

1. *Timed Process*: Only applicable temporal property for a process is *deadline*.
2. *Non-time Action*: The action with no temporal properties is same as the one with the temporal properties of $[0,-,1,-]$.
3. *Empty*: Only applicable temporal property for the *Empty* action is *execution time*.

4.5. Characteristics

The temporal properties are directly specified to each action and process in dT-Calculus. The specification of the temporal properties for both actions and processes allows the temporal requirements for both actions in a processes and the process itself to be specified and analyzed at the same time.

The introduction of the periodic temporal property has many advantages than other process algebras in specification of different types of repeating processes. Generally, the starting time of each synchronous action depends on the ready time of its partner action so that the same actions may require different total execution or termination time of their synchronous actions.

$P_{[r,to,e,d]} = P_{[-,-,-,d]}$	Timed Process
$A = A_{[0,-,1,-]}$	Non-time Action
$\emptyset_{[r,to,e,d]} = \emptyset_{[-,-,e,-]}$	Empty

Table 2. Temporal Laws of dT-Calculus.

That is, there is some problem of not being able to specify explicitly and precisely the temporal properties of periodic actions in the following form:

$$A \cdot \emptyset_{[-, -, e, -]} \cdot A \cdot \emptyset_{[-, -, e, -]} \cdot A \cdot \emptyset_{[-, -, e, -]} \cdot \dots \quad (6)$$

It is intended to specify the above periodic actions with empty actions, but the empty actions with fixed execution time are not appropriate because their interaction times for synchronization can be different from each other. However, there is an advantage that there is no need to consider such time for synchronous interactions if the periodic temporal property is used. The specification of the periodic requirements becomes very simple since the next execution of an action will be performed after elapsing the periodic temporal duration without calculating the temporal length left over up to the next re-execution of the action following the immediate execution of the action.

4.6. Graphical representation

There are two graphical representations for dT-Calculus: system view and process view. *System view* represents graphical relationships among processes in a system: containment and




Icon	
Process	
Channel	
Movement	

Table 3. Icon for system view.




















Icon							
Process Lane		Start		End		Other Process	
Exit		Choice		Parallel		Send	
Receive		Empty		InR		OutR	
GetR		PutR		InP		OutP	
GetP		PutP		Sequence			

Table 4. Icon for process view.

interactions. *Process view* represents graphical relationships among actions in a process: precedence and control flow. These views show in-the-large (ITL) view of a system and in-the-small (ITS) views for its processes, respectively. And they provide better understanding of the system and the processes in the visual representation. **Tables 3** and **4** show the icons for the views, respectively.

5. Example

This section describes the specification of a distributed mobile real-time IoT system in dT-Calculus with a *Smart Emergency Evacuation System* (SEES) example.

SEES is a system that activates evacuation plan with supporting devices in buildings or facilities, in case of fire or threat, by detecting the source of fire or threat as well as the people and their movements in the building, and guiding them safely out of the building until all of them move out of the building safely in both active or passive manner [10].

5.1. Requirements

SEES needs a set of secure requirements since it guarantees safe evacuation of people in a building in case of fire or threat. The requirements include, as stated, provision of the evacuation plan, detection of the source of fire or threat as well as the people and their movements in the building, automatic notification of the fire and threat to police and 911, and safe guidance of the residents out of the building. It can be summarized as follows:

1. Req 1: Sensors must confirm occurrence of fire or threat continuously.
2. Req 2: Controller must send fire or threat alarm to all the people in case of fire or threat.
3. Req 3: Controller must guide all the people to the safe areas without fire in both present and near future.
4. Req 4: The evacuation of all the people must be completed in 25 time units.
5. Req 5: 911 must evacuate the people who are not escaped from the fire.

In case that these requirements are not satisfied, it is possible for people not to escape from fire or to escape through insecure paths, causing loss of human lives. Therefore it is very important to specify these requirements formally and to verify their satisfiability.

5.2. Specification

As shown in **Figure 10** in dT-Calculus, the SEES in the example operates as follows:

1. A fire is detected by sensor(s), and is informed to the controller by the sensor(s).
2. The controller informs the people in the building of the fire or threat, and, at the same time, shows the evacuation paths as planned.

```

Sys := Building[ControlSystem|StairA|SensorA||StairB|SensorB] |1st floor|2nd floor|[P1|P2]] |911;
Control System := (CS(FireA) · P1(StairB) · P2(StairB) + CS(FireB) · P1(StairA) · P2(StairA))
                · CE(Call) · CS(P1)[0,0,1,7] \ CE(P1) · CS(P2)[0,0,1,4] \ CE(P2);
SensorA := ((SA(Fire)[0,3,1,0] · CS(FireA)) \ 03)6,∞;
SensorB := ((SB(Fire)[0,3,1,0] · CS(FireB)) \ 03)6,∞;
P1 := (P1(StairB) · (∅.RC(P1).out 2nd.out Building + out 2nd.in StairB.out StairB.in 1st.out 1st.out Building))
    + (P1(StairA) · (∅.RC(P1).out 2nd.out Building + out 2nd.in StairA.out StairA.in 1st.out 1st.out Building));
P2 := (P2(StairB) · (∅.RC(P2).out 2nd.out Building + out 2nd.in StairB.out StairB.in 1st.out 1st.out Building))
    + (P2(StairA) · (∅.RC(P2).out 2nd.out Building + out 2nd.in StairA.out StairA.in 1st.out 1st.out Building));
StairA := (P1 in[0,0,1,10] · P1 out) \ 0 · (P2 in[0,0,1,10] · P2 out) \ 0;
StairB := (P1 in[0,0,1,10] · P1 out) \ 0 · (P2 in[0,0,1,10] · P2 out) \ 0;
1st floor := (P1 in[0,0,1,10] · P1 out) \ 0 · (P2 in[0,0,1,10] · P2 out) \ 0;
2nd floor := P1 out[0,0,1,10] \ 0 · P2 out[0,0,1,10] \ 0 · (911 in[0,0,1,20] · P1 out[0,0,1,5] \ 0 · P2 out[0,0,1,5] \ 0 · 911 out) \ 0;
Building := (SA(Fire) + SB(Fire)) · (P1 out[0,0,1,13] · CS(P1)) \ 0 · (P2 out[0,0,1,13] · CS(P2)) \ 0
            · (911 in[0,0,1,10] · P1 out[0,0,1,5] \ 0 · P2 out[0,0,1,5] \ 0 · 911 out) \ 0;
911 := Ce(Call) · (CE(P1) · in Building · in 2nd · RC(P1) · out 2nd · out Building
                + CE(P2) · in Building · in 2nd · RC(P2) · out 2nd · out Building
                + CE(P1) · CE(P2) · in Building · in 2nd · RC(P1) · RC(P2) · out 2nd · out Building)[0,10,0,0] \ 0;

```

Figure 10. The SEES example in dT-Calculus.

3. The controller tracks all the people in the building while they are evacuating, and informs the current status of the evacuation to 911 in real-time, so that the people trapped in the building can be monitored in real-time as planned.
4. 911 rescues the people trapped in the building in order, based on the status of the fire or threat in the building and the availability of the rescue facilities and devices.

In the specification, the following actions have been declared in Process *Building* and Process *Control System* to detect the case that the people cannot be evacuated from building autonomously:

$$Building ::= \dots P1 \text{ out}_{[0,0,1,14]} \cdot CS(\overline{P1}) \dots \quad (7)$$

$$Control \ System ::= \dots CS(P1)_{[0,0,1,7]} \setminus CE(\overline{P1}) \quad (8)$$

The above code implies that, when *P1* moves out of the building, it sends *CS* a signal of its safe evacuation, and that, if not, that is, if the signal is not received in the deadline of 7 time units of $[0,0,1,7]$ by *CS*, the non-evacuation situation of *P1* is informed to 911 by the exception handler process *CE* of *CS*.

In the specification from **Figure 10**, sensors, *SensorA*, and *SensorB*, are defined to perform their actions in repetition by the period properties of dT-Calculus: normally their fire alarm actions do not occur by timeout in normal case of no fire, however, in case of fire, they have to occur in order to inform *Control System* of the fire.

There are two people in the building and there are two choices for them in case of fire: one for evacuation safely from the building, and another for non-evacuation.

5.3. Graphical representation

The textual specification in dT-Calculus can be represented graphically in two views: *in-the-large* (ITL) and *in-the-small* (ITS). The ITL view can be considered as *system view* consisting of processes interacting together with communication and movements. The ITL view can be considered as *process view* with the detailed actions. **Figure 11** shows the ITL view of the SEES example, and **Figures 12** and **13** show the ITS views of the processes in the example.

In order to construct the ITL view for the example, it is necessary to understand main processes and their containment relations from the example, which is textually specified with dT-Calculus in Section 5.2 as follows:

$$\text{Sys} := \text{Building}[\text{Control System} | \text{StairA}[\text{SensorA}] | \text{StairB}[\text{SensorB}] | \text{1st floor} | \text{2nd floor}[P1 | P2]] | 911; \quad (9)$$

In **Figure 11**, $P1$ and $P2$ are placed in *2nd floor* since they are defined as contained processes of *2nd floor* in Eq. 9. Similarly, *SensorA* and *SensorB* are placed in *StairA* and *StairB*, respectively, in the figure, since they are defined as contained process of *StairA* and *StairB*, respectively, in the equation. Further *1st floor*, *2nd floor*, *StairA* and *StairB* are placed in *Building* in the figure, since they are defined as contained processes of *Building* in the equation. However 911 is placed outside of *Building* in the figure since it is defined as a parallel process of *Building* in the equation. In addition, the edges in the view are the channels for communication among the processes in the example.

In order to construct the ITS view of each process as shown in **Figures 12** and **13**, it is necessary to understand the types of actions in each process and their order of execution. For example, **Figure 14** shows the ITS view of *Building* from **Figure 13**. The figure shows actions as nodes

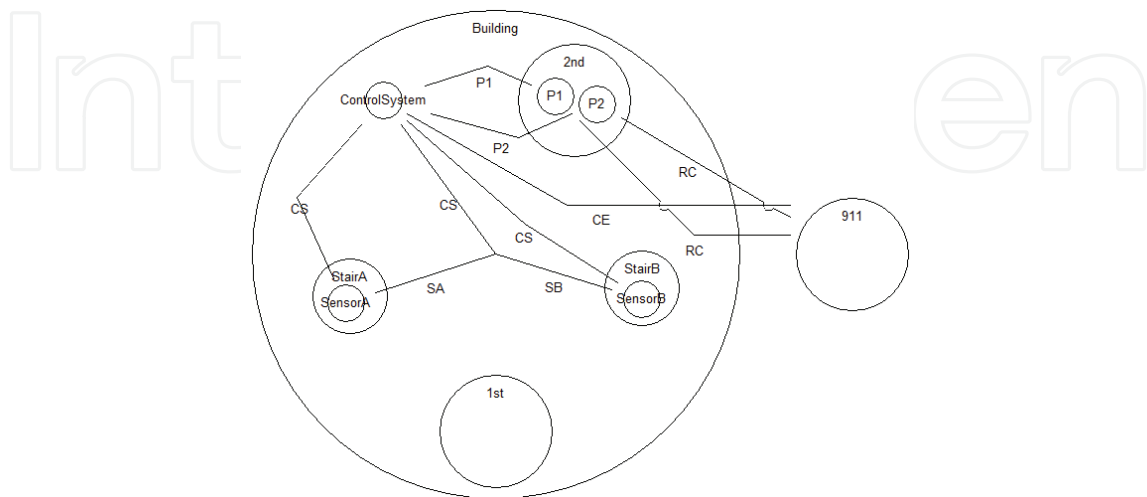


Figure 11. ITL view of the SEES example.

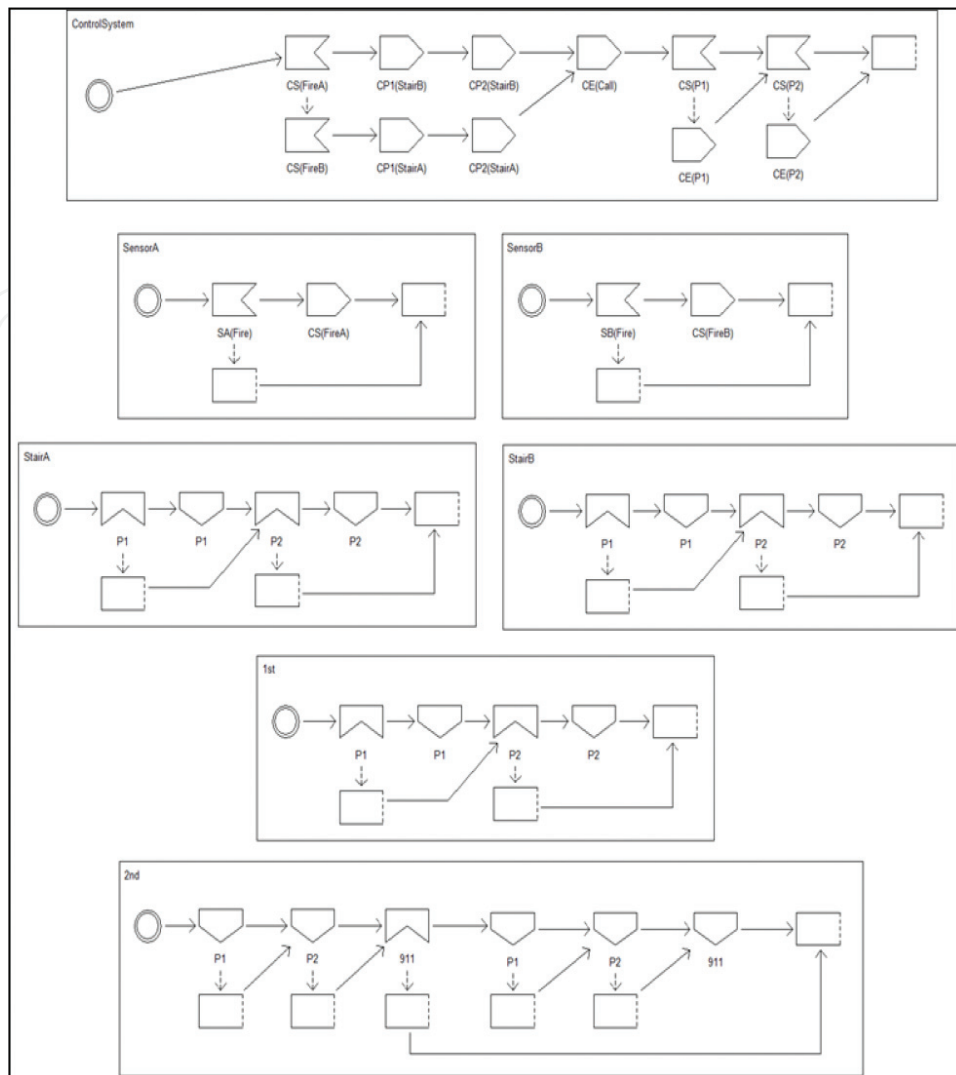


Figure 12. ITS views of the SEES e2example (1).

and their execution order as directed edges for *Building*, which is textually specified with dT-Calculus in Section 5.2 as follows:

$$\begin{aligned}
 \text{Building} := & (SA(\overline{\text{Fire}}) + SB(\overline{\text{Fire}})) \cdot (P1 \text{ out}_{[0,0,1,13]} \cdot CS(\overline{P1})) \setminus \emptyset \cdot (P2 \text{ out}_{[0,0,1,13]} \cdot CS(\overline{P2})) \setminus \emptyset \\
 & \cdot (911 \text{ in}_{[0,0,1,10]} \cdot P1 \text{ out}_{[0,0,1,5]} \setminus \emptyset \cdot P2 \text{ out}_{[0,0,1,5]} \setminus \emptyset \cdot 911 \text{ out}) \setminus \emptyset;
 \end{aligned}
 \tag{10}$$

Building performs the $SA(\overline{\text{Fire}}) + SB(\overline{\text{Fire}})$ first. The *Choice* operation in the action is graphically represented with its *Choice* icon in the figure, including its two independent execution paths. And it is followed by a sequence of timed actions with exception, represented by their graphical icons. Firstly, $(P1 \text{ out}_{[0,0,1,13]} \cdot CS(\overline{P1})) \setminus \emptyset$ is graphically represented by a pair of ordered action of $P1 \text{ out}_{[0,0,1,13]}$ and $CS(\overline{P1})$ with its exception, that is, \emptyset , in the figure. Other timed actions are similarly represented in the same graphical pattern.

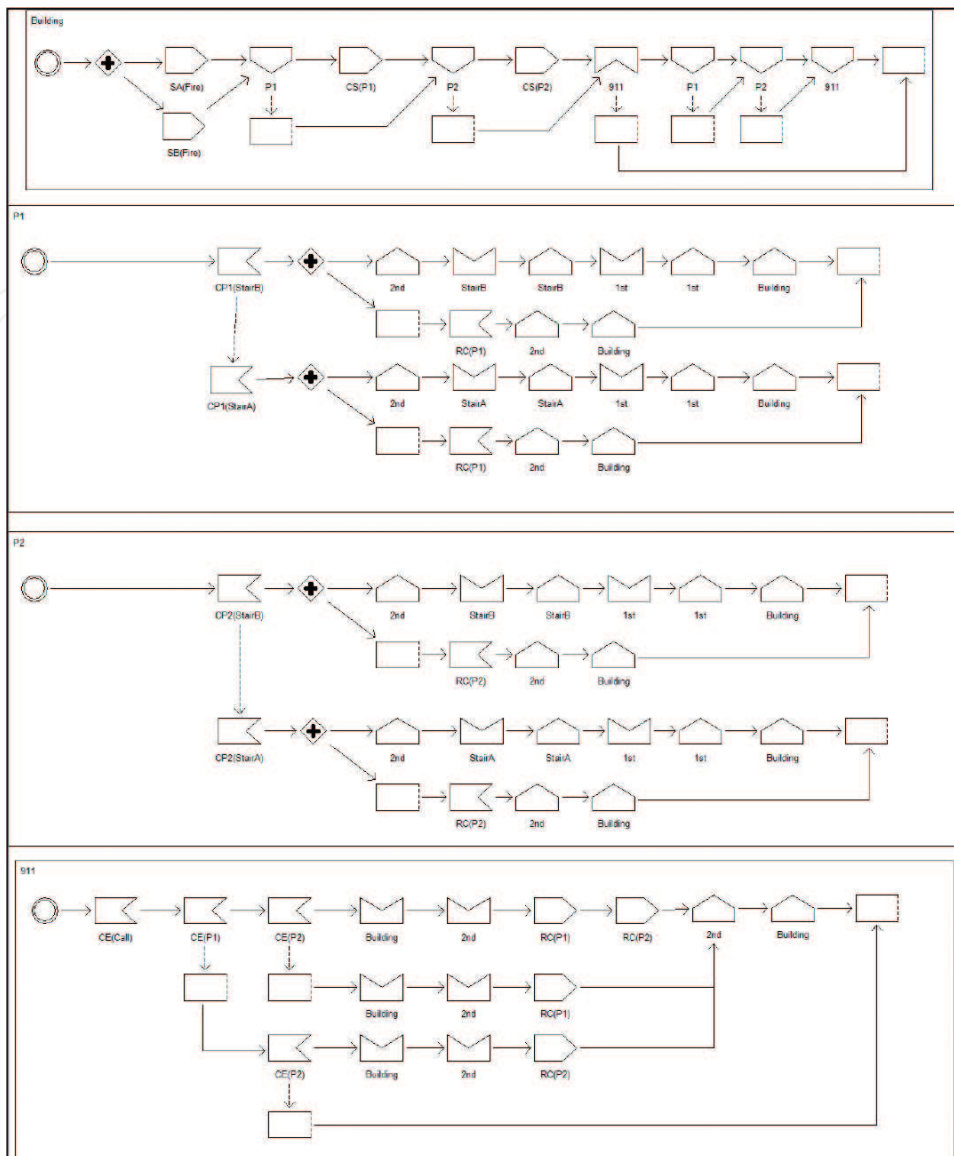


Figure 13. ITS views of the SEES example (2).

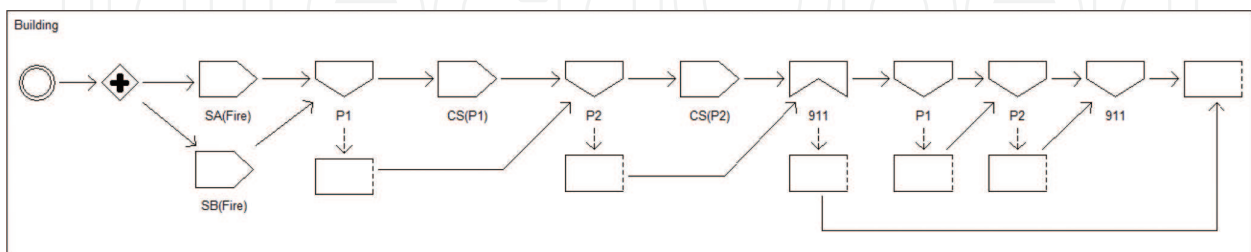


Figure 14. ITS view of the building process.

5.4. Execution

Figure 15 shows the execution model for the SEES example. It consists of total 8 execution paths. Note that an execution path implies each independent case of execution by the example.

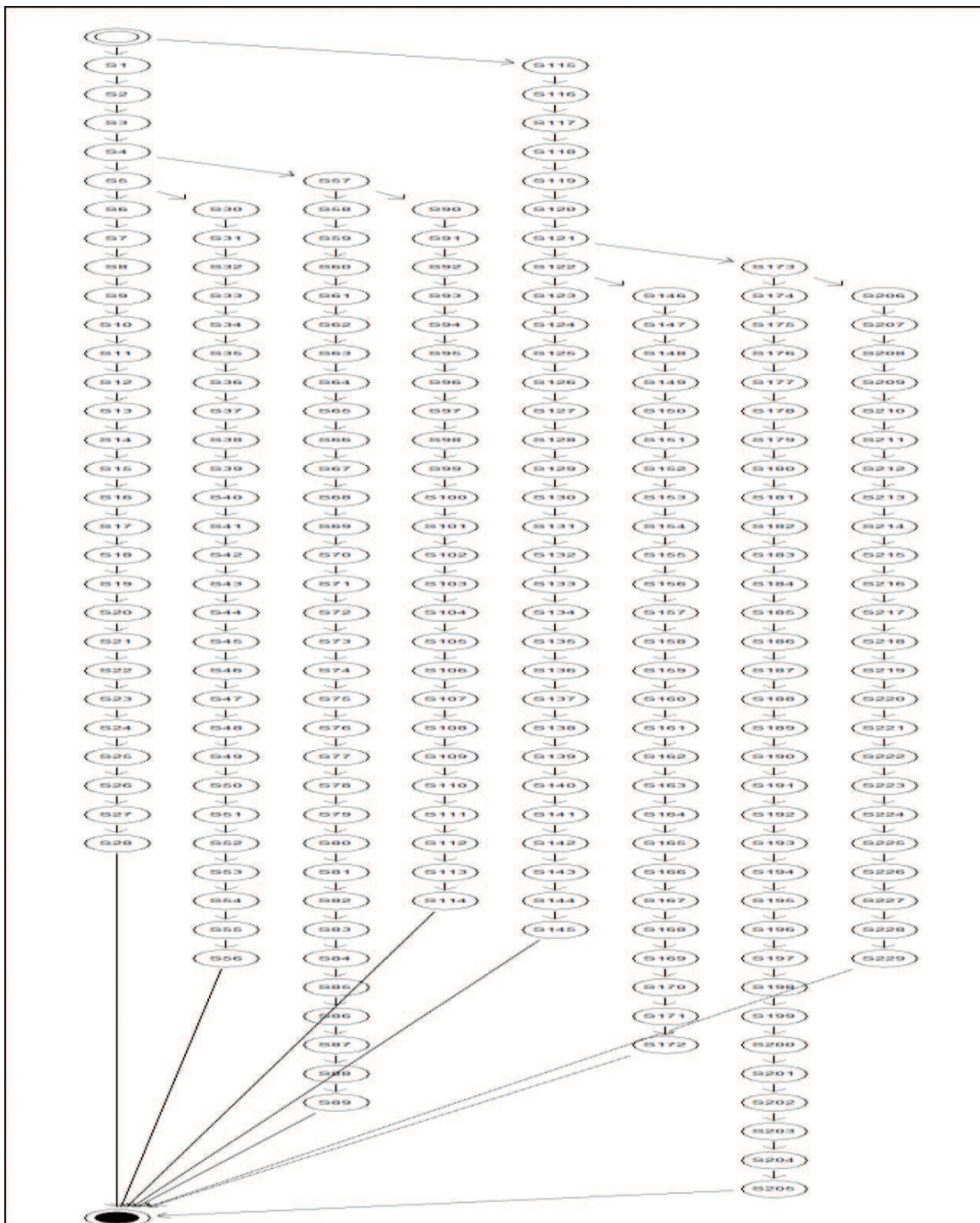


Figure 15. Execution paths of the SEES system.

It can be obtained by analyzing all the possible synchronization cases in the example. The icons in the model are defined in **Table 5**.

As shown in **Figure 15**, there are total eight paths: two locations for fire, two cases of evacuation for two persons, and consequently eight cases in total.

Firstly, for each execution path of successful evacuation, it is possible to perform analysis of their temporal properties as follows. Let us consider the case that a fire occurs at Stairs A:






Icon	
Start	
End	
State	
Deadlock	
Sequence	

Table 5. Icon for execution model.

1. T1: A fire occurs at Stairs A.
2. T2: A sensor detects the fire and informs a controller of the fire with a signal.
3. T3: The controller informs *P1* on Floor 2 of an evacuation path through Stairs B.
4. T4: The controller informs *P2* on Floor 2 of an evacuation path through Stairs B.
5. T5: The controller informs 911 of the fire, and *P1* enters Stairs B.
6. T6: *P2* enters Stairs B.
7. T7: *P1* enters Floor 1.
8. T8: *P2* enters Floor 1.
9. T9: *P1* moves out of the building.
10. T10: *P2* moves out of the building, and the controller detects that *P1* moved out of the building.
11. T11: The controller detects that *P2* moved out of the building.

All the people moved out of the building in 10 time units. And the controller detected their evacuation in 11 time units. Since there are more actions left to be performed by 911, it takes more time units for the system to terminate its own mission.

Secondly, for each execution path of failed evacuation, it is also possible to perform analysis of their temporal properties as follows. Let us consider the case that a fire occurs at Stairs A:

1. T1: A fire occurs at Stairs A.
2. T2: A sensor detects the fire and informs a controller of the fire with a signal.
3. T3: The controller informs *P1* of an evacuation path through Stairs B.

4. T4: The controller informs $P2$ of an evacuation path through Stairs B , and $P1$ is not able to move of out Floor 2.
5. T5: The controller informs 911 of the fire.
6. T6: $P2$ enters Stairs B .
7. T8: $P2$ enters Floor 1.
8. T10: $P2$ moves out of the building.
9. T12: The controller detects that $P1$ is still on Floor 2.
10. T13: The controller informs 911 of the non-evacuation of $P1$.
11. T14: The controller detects that $P2$ moved out of the building, and 911 moves into the building to rescue $P1$.
12. T16: 911 finds $P1$ and provides the first treatment.
13. T18: $P1$ moves out of the building.
14. T19: 911 moves out of the building.

For evacuation, $P2$ takes 10 time units, but $p1$ takes 18 time units due to rescue time required for 911 to handle $P1$'s non-evacuation situation. Once all the people are safely evacuated, the system will terminate its mission. However it will takes little more time due to some left-over actions by 911.

As a result of analysis, it can be confirmed that, in case of the fire at Stairs A , all the people were evacuated safely in 20 time units. Similar to the case of the fire at Stairs A , it can be confirmed that, in case of the fire at Stairs B , all the people were evacuated safely in 20 time units. Consequently it can be concluded that all the people in the building will be safely evacuated in time in any case of fires.

5.5. Analysis

In order to assure the safety of SEES, it is necessary to verify if the safety requirements, specified in dT-Calculus, in Section 5.1, are satisfied or not. All the five requirements specified in the section must be verified in order to prevent loss of lives from happening by fire as follows:

1. Req 1: Sensors must confirm occurrence of fire continuously.

It is specified in the SEES specification for $SensorA$ and $SensorB$ as follows. They are detecting fires in the same actions in different locations, that is, A and B :

$$SensorA := \left(\left(SA(\overline{Fire})_{[0,3,1,0]} \cdot CS(\overline{FireA}) \right) \setminus \emptyset_3 \right)^{6,\infty} \quad (11)$$

Each sensor performs a fire-detecting action for 3 time units. In case of no fire, it terminates its action immediately, but it repeats its fire-detecting action repeatedly as the following periodic actions with the specifier of its “6,∞.” However, in case of fire, it notifies the fire to the controller, and similarly, it repeats its fire-detecting action repeatedly as the following periodic actions.

2. Req 2: Controller must send a fire alarm to all the people in case of fire or threat.

The controller performs the following actions in case of fire:

$$(CS(FireA) \cdot P1(\overline{StairB}) \cdot P2(\overline{StairB}) + CS(FireB) \cdot P1(\overline{StairA}) \cdot P2(\overline{StairA})) \quad (12)$$

No matter where the fire occurs, it can be verified that the alarm is sent to all the people in the building: $P1$ and $P2$.

3. Req 3: Controller must guide all the people to the safe areas without fire in both present and near future.

In the actions in 2), it can be verified that the people receiving the $FireA$ by CS get the $StairB$ signal for evacuation and, similarly, that the people receiving the $FireB$ by CS get the $StairA$ signal for evacuation. It guarantees that the people in the fire areas are evacuating through the non-fire areas.

4. Req 4: The evacuation of all the people must be completed in 25 time units.

As shown in Section 5.4, the autonomous evacuation, that is, the evacuation of the people without 911, takes 10 time units. However the heteronomous evacuation, that is, the evacuation of the people by 911, takes little longer than the autonomous case, since it requires the time that 911 arrives at the site. In this case, the controller has to recognize the situation of non-evacuation of the people at $T12$ and $T17$, and 911 has to evacuate the people at $T21$ and $T22$. Finally, $P1$ is evacuated at $T1$, and $P2$ is evacuated at $T24$. In both cases, it can be verified that all the people are evacuated in 25 time units.

5. Req 5: 911 must evacuate all the people who are not escaped from the fire.

911 performs the following actions after the call:

$$(CE(P1) \cdot \dots + CE(P2) \cdot \dots + CE(P1) \cdot CE(P2) \cdot \dots) \quad (13)$$

It shows that the evacuations are performed by the signals from the controller, as the following calls for the signals of the controller show:

$$\dots \cdot CS(P1)_{[0,0,1,7]} \setminus CE(\overline{P1}) \cdot CS(P2)_{[0,0,1,4]} \setminus CE(\overline{P2}) \quad (14)$$

$CS(P1)$ and $CS(P2)$ are the signals from the people when they are evacuating from the building. In case that the signals are not transmitted to the controller in the certain period of time, the controller sends 911 the non-escaping signal to indicate the non-evacuation situation of the

people. It is be verified that SEES guarantees that the controller recognizes all the non-evacuated people in the building and informs 911 of the situations, and that 911 evacuates them in time.

6. Comparative analysis

6.1. Main characteristics of IoT systems

The main characteristics of the IoT-based systems are shown in many literatures [11–13]. These can be summarized as follows with respect to process algebra:

1. **Mobility:** A number of devices in the systems are able to move their positions in geographical space. The devices should be able to get IoT services at any place and environment.
2. **Real-time:** The IoT devices in the systems should be able to get IoT services in real-time.
3. **Interactivity:** The interactions among the IoT devices in the systems must be possible, i.e., communication among the electronic devices in the smart home.

Especially, distributed mobile real-time IoT systems must have the above characteristics in order to operate properly in real-time without faults over geographical space with temporal restrictions.

6.2. Timed pi-Calculus

Timed pi-Calculus is a process algebra that is designed to specify and analyze mobile services. Timed pi-Calculus is the timed version of pi-Calculus, which allows time-stamp and clock be passed additionally during value passing: the temporal requirements of the process movements can be specified. However there is a limitation that the execution time of an action cannot be specified directly on the action. Further it is difficult to analyze the execution time, the deadline, and others of an action, since such temporal properties are represented by the passing time-stamp and clock. Similarly the movement in the algebra is inappropriate to represent a real movement of a process since it is represented by value passing. Consequently such indirect representation of a movement may result in distortion of the patterns of real movements since the representation reduces the scope of the possible movements in expression.

6.3. Timed Mobile Ambient

Timed Mobile Ambient is a process algebra that allows specification of temporal requirements by adding temporal properties on the existing movements of processes from Mobile Ambient. Temporal properties are added to process movements controlled by capabilities, and the process with the properties performs as follows: if the process performs an action within the valid time, it performs normally as in Mobile Ambient. If not, the existing process is intentionally terminated and a safe process is executed instead, in order to handle this abnormal

situation. Timed Mobile Ambient solves the incapability of temporal specification of Mobile Ambient, but it is difficult to reason about starting time of processes since there is no other temporal properties except deadline. In addition, it is difficult to understand intuitively process synchronization since the synchronization is represented by the movements of ambients.

6.4. d-Calculus

d-Calculus is a process algebra that is designed to express direct process movements into or out of other processes both autonomously and heteronomously. It allows various types of mobile requirements to be specified, but only a simple type of temporal requirements for process movements is possible: a temporal bound of the minimum and maximum limits. It results in limited specification of the temporal requirements of the movements as well as analysis of the requirements. In addition, specification can be represented in both text and graph in order to increase visibility of the specification as well as comprehensibility. However there are limitations in specification of temporal properties: the execution time is only possible for an action and deadline is specified only by exception. It implies that only simple temporal specification is possible, but complex temporal specification for the smart EMS example is not allowed.

6.5. dT-Calculus

However, dT-Calculus overcomes these limitations of these algebras. Since it is an extension version of d-Calculus, it can utilize all different types of direct movements of processes. Besides, it is possible to specify complex temporal requirements of the smart mobile service by supplying a variety of additional temporal properties. Further, the analysis of the temporal properties is relatively easy since the properties are directly specified on actions and processes. And it is possible to specify exceptional handling to solve errors or faults caused by any violation of timeout and deadline.

6.6. IoT-based comparison

The first three process algebras can be analyzed with dT-Calculus with respect to the IoT characteristics stated in Section 6.1, as follows:

1. **Mobility:** A number of IoT devices are moving around in the IoT systems in a various manners. For example, a device containing other devices can move in and out of other devices, autonomously or heteronomously. In Timed pi-Calculus, the movements of processes can be expressed with value passing only. Consequently there are limitations to express various kinds of direct movements. In Timed Mobile Ambient, there are three in, out, open movement actions. However there is no movement action for passive or heteronomous movement. In d-Calculus and dT-Calculus, it is possible to express both autonomous and heteronomous movements of processes since they provide both the active actions of in, out and the passive actions of get, put.
2. **Real-time:** The IoT systems should provide their services in real-time. It means that the process algebras for the systems must have capability to express real-time properties of the

Process Algebra	IoT Characteristic		
	Types of movement Properties	Types of temporal properties	Interactions
Timed pi-Calculus	Indirect movements	Deadline	Communication
Timed Mobile Ambient	in, out, open	Deadline	Communication Movements
d-Calculus	in, out, get, put	Execution time Deadline	Communication Movements
dT-Calculus	in, out, get, put	Ready time Time out Execution time deadline period	Communication Movements

Table 6. Comparison of dT-Calculus with other algebras by the IoT characteristics.

services. In Timed pi-Calculus, it is possible to specify the temporal properties of processes by providing time-stamp and clock through value passing. But it is not possible to specify execution time of its actions. In Timed Mobile Ambient, it is possible to specify only temporal property of deadline for process movement with capability, but other properties are not possible. In d-Calculus, only execution time and deadline properties are possible, but others are not possible. However, in dT-Calculus, other properties, such as, ready time and time out, are possible, beside execution time and deadline properties of d-Calculus.

3. Interactions: All the devices in the IoT system should interact together. It implies that the process algebras for the systems must have capability to express the interactions. All the above algebras are able to express interactions among processes, but there are differences in the types of the interactions. In Timed pi-Calculus, the interactions are based on of synchronized communication. In Time Mobile Ambient, the interactions are based on capability-based movements, besides communication. In d-Calculus and dT-Calculus, the interactions are based on both communication and movements by synchronization.

Table 6 shows the summary of the analysis with respect to the IoT characteristics.

7. SAVE

In order to demonstrate the feasibility of the approach in the paper, a tool, called SAVE (Specification, Analysis, Verification and Evaluation) [14], has been developed on the ADOxx meta-modeling platform [15]. As shown in **Figure 16**, it consists of four basic components as follows:

- Modeler: It provides capability to specify system and process views.
- EM Generator: It generates an execution model (EM) for the views and makes each path of the model to be selected for simulation.

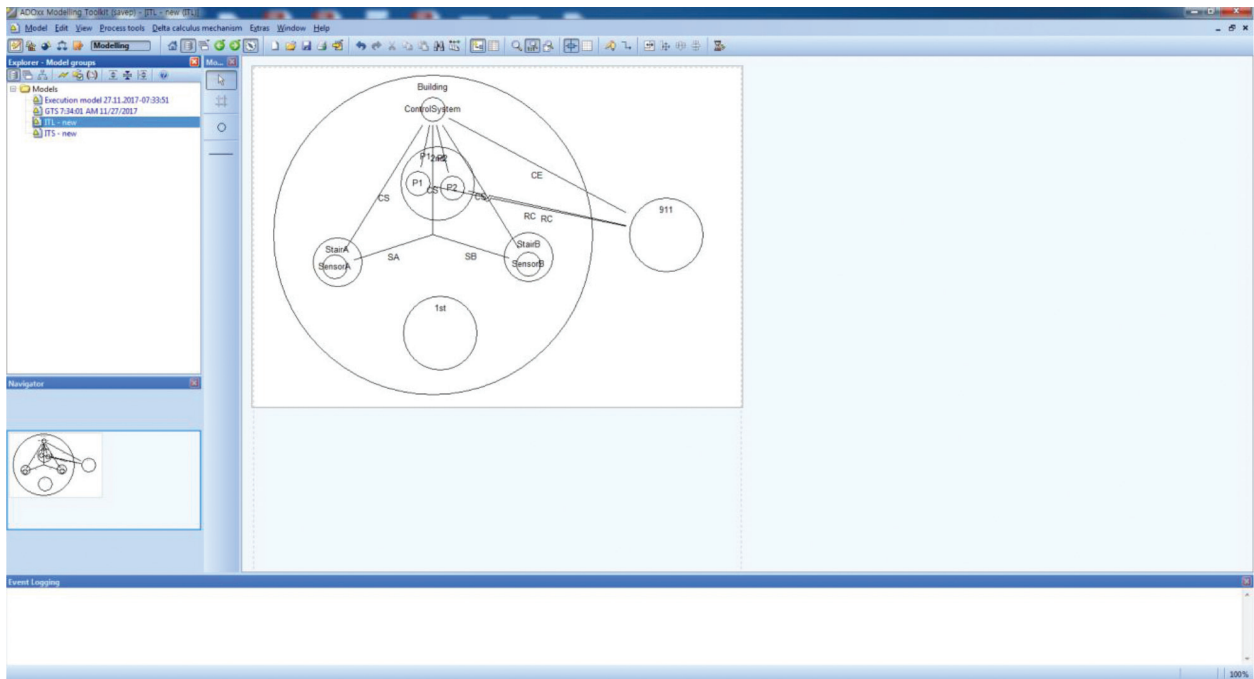


Figure 17. ITL model in SAVE.

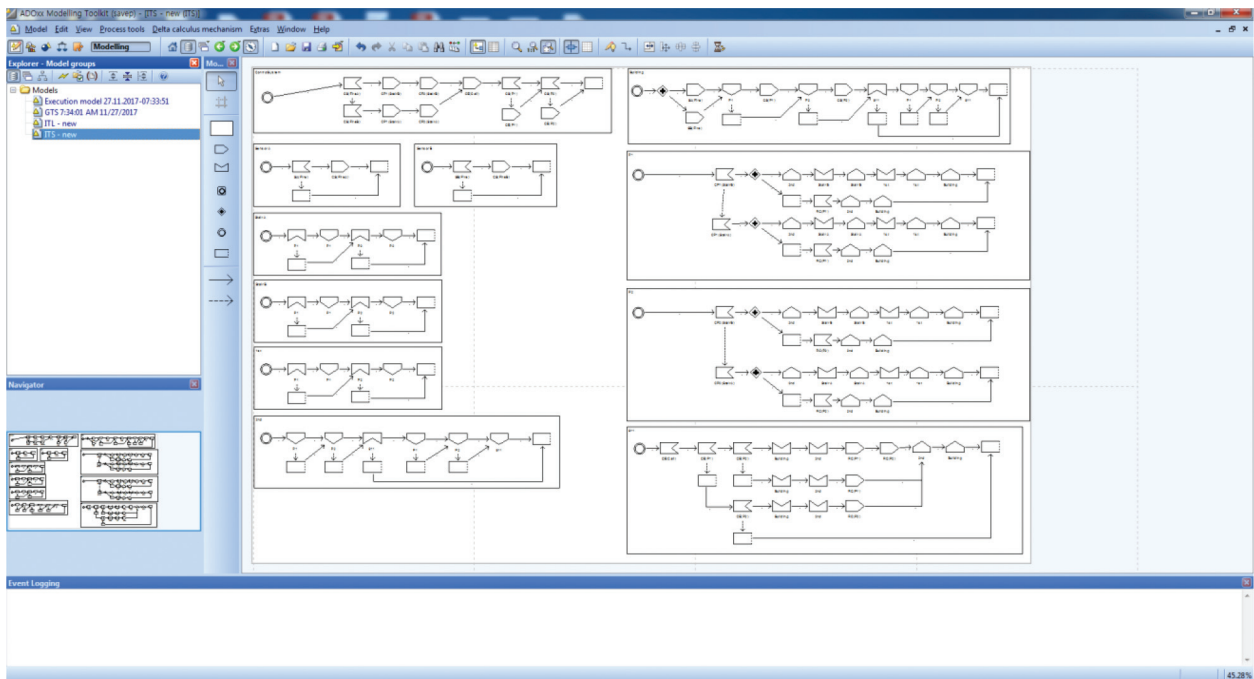


Figure 18. ITS models in SAVE.

Based on the simulation model, it is possible to analyze and verify the temporal requirements of IoT systems. **Figure 20** shows the simulation model for the first path of the SEES example in **Figure 19**.

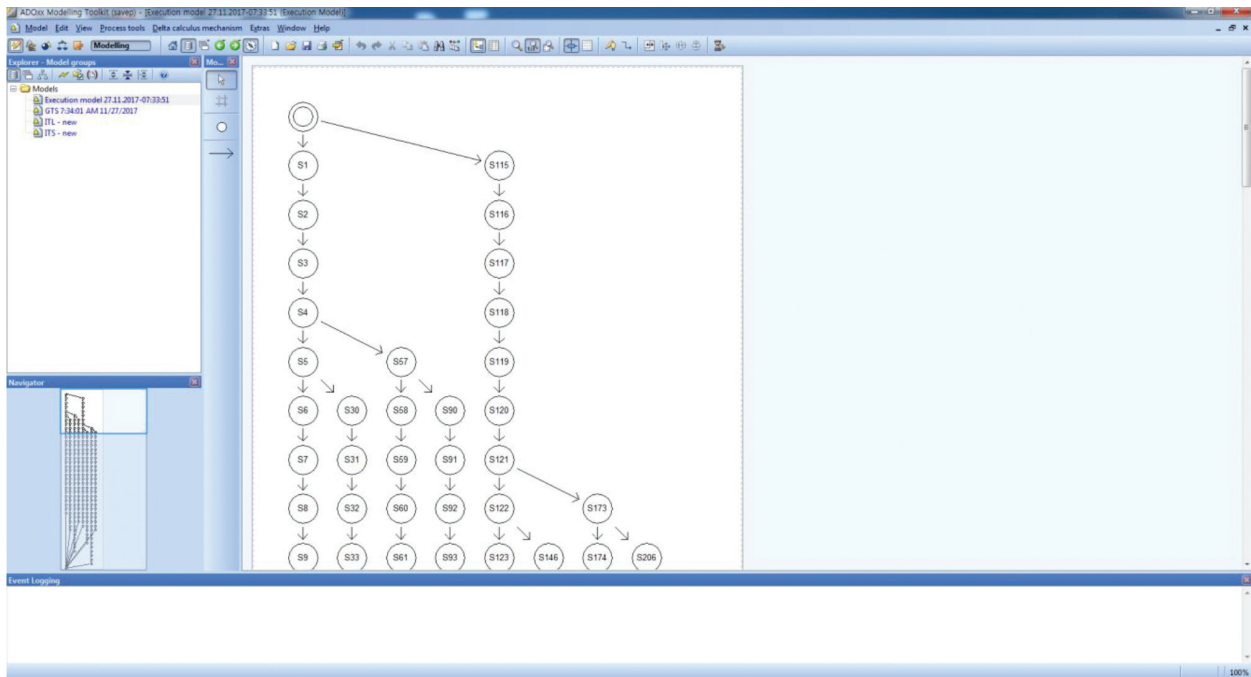


Figure 19. Execution model in SAVE.

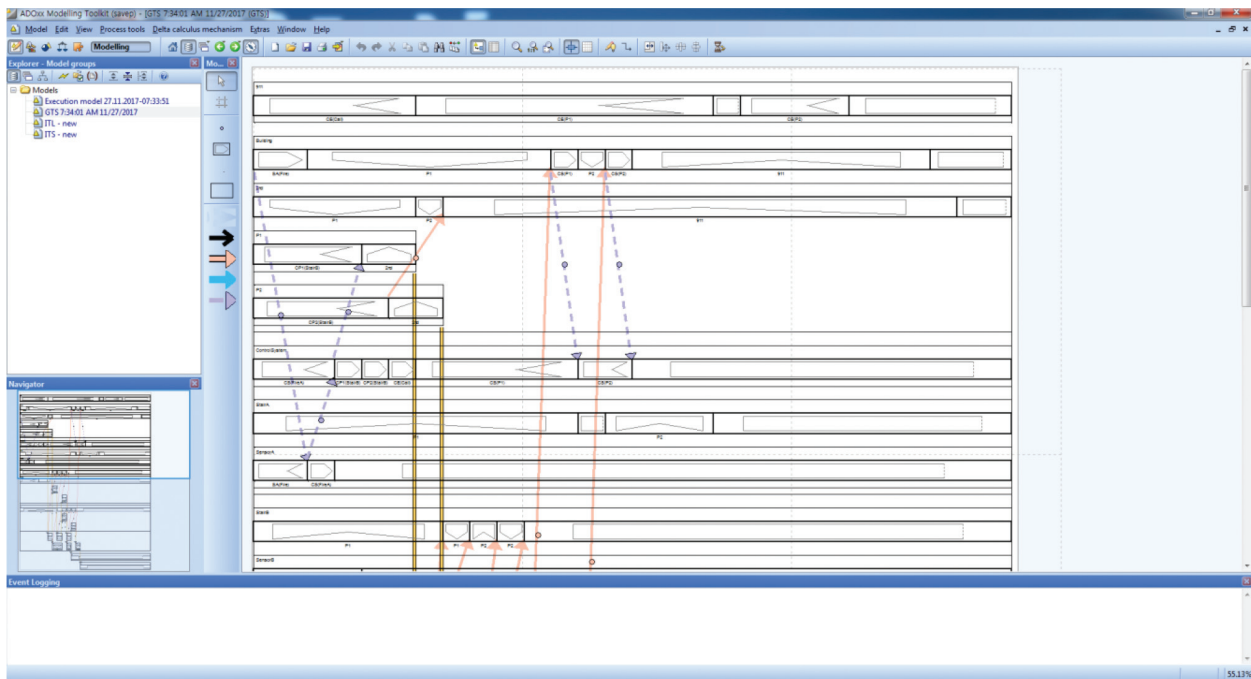


Figure 20. Simulation model in SAVE.

8. Conclusions and future research

This paper proposed dT-Calculus for mobile and temporal specification of the distributed mobile real-time IoT systems. The algebra extended d-Calculus for specification and analysis

of a variety of different types of temporal properties at the direct movement actions and the mobile processes. Further a tool, called SAVE, has been developed to demonstrate the feasibility of the approach with the algebra.

In the paper, the process algebra for specification with temporal properties is presented. In the future research, the different types of verification methods are developed to demonstrate the usability of dT-Calculus, based on a logic system, including SAVE with a verification model.

Acknowledgements

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 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).

Author details

Sunghyeon Lee, Yeongbok Choe and Moonkun Lee*

*Address all correspondence to: moonkun@jbnu.ac.kr

Chonbuk National University, Jeonju-si Jeonbuk, Republic of Korea

References

- [1] Chen C-Y, Hasan M, Mohan S. Securing Real-Time Internet-of-Things. arXiv preprint arXiv. 2017;1705:08489
- [2] Choe Y, Lee M. Algebraic method to model secure IoT. In: Karagiannis D, Mayr HC, Mylopoulos J, editors. Domain-Specific Conceptual Modeling. Switzerland: Springer International Publishing; 2016. pp. 335-355. Ch. 15
- [3] Saeedloei N, Gupta G. Timed π -Calculus. Trustworthy Global Computing. Lecture Notes in Computer Science. Vol. 8358. Cham: Springer. 2014
- [4] Milner R, Parrow J, Walker D. A calculus of mobile processes (i–ii). Information and Computation. 1992:1-77
- [5] Aman B, Ciobanu G. Mobile ambients with timer and types. In: International Colloquium on Theoretical Aspects of Computing. Berlin Heidelberg: Springer; 2007
- [6] Cardelli L, Gordon A. Mobile ambients. In: Nivat M, editors. ETAPS 1998 and FOSSACS 1998. LNCS. Vol. 1378. Heidelberg: Springer; 1998. pp. 140-155

- [7] Choe Y, Lee M. δ -Calculus: Process algebra to model secure movements of distributed mobile processes in real-time business application. In: 23rd European Conference on Information Systems; 2015
- [8] Choe Y, Choi W, Jeon G, Lee M. A tool for visual specification and verification for secure process movements. In: eChallenges e-2015; 2015
- [9] SAVE tool. Available from: <http://austria.omilab.org/psm/content/save/info>
- [10] Snoonian D. Smart buildings. IEEE Spectrum. 2003;**40**(8):18-23
- [11] Chung S-m, Choi J-h, Park J-w. Design of software quality evaluation model for IoT. Journal of the Korea Institute of Information and Communication Engineering. 2016; **20**(7):1342-1354
- [12] Liu Y, Zhou G. Key technologies and applications of internet of things. 2012 Fifth International Conference on Intelligent Computation Technology and Automation (ICICTA); IEEE; 2012
- [13] Sarkar C, et al. A scalable distributed architecture towards unifying IoT applications. 2014 IEEE World Forum on Internet of Things (WF-IoT); IEEE; 2014
- [14] Choe Y, Lee S, Lee M. SAVE: An environment for visual specification and verification of IoT. 2016 IEEE 20th International on Enterprise Distributed Object Computing Workshop (EDOCW); IEEE; 2016
- [15] Fill H, Karagiannis D. On the conceptualisation of modeling methods using the ADOxx meta modeling platform. Enterprise Modeling and Information Systems Architectures. 2013;**8**(1):4-25