Process Algebra to Model Timed Movements of Processes in Distributed Mobile Real-Time Systems

YEONGBOK CHOE, SUNGHYEON LEE and MOONKUN LEE
Chonbuk National University
567 Beakje-dearo Deokjin-gu
Jeonju-si Jeonbuk 54896, REPUBLIC OF KOREA
e-mail:moonkun@jbnu.ac.kr

Abstract: - There are strong needs for both mobile and temporal properties in process algebras designed to specify and analyze distributed mobile real-time systems (DMRS). However there are some limitations in those algebras to specify both movements and temporalness at the same time, such as, Timed pi-Calculus and d-Calculus, as follows: 1) Timed pi-Calculus cannot specify both the execution time of action and movements directly, and 2) d-Calculus can specify only a simple pattern of temporal conditions, that is, the lower and upper bounds of the execution time. In order to solve these limitations, this paper proposes dT-Calculus with expressive power of movements of processes with a number of temporal conditions. dT-Calculus extended the basic temporal properties of synchronous process movements of d-Calculus into more specific temporal properties in order to specify and analyze the temporal property of DMRS more effectively: *ready time*, *execution time*, *waiting time*, *deadline*, etc. In order to simulate the proposed process movement with temporal properties, the SAVE tool has been developed on the ADOxx Meta-Modeling Platform and demonstrates efficiency and effectiveness of the proposed approach with an EMS example.

Keywords: dT-calculus; d-calculus; Process Algebra; Mobility; Time; SAVE

1 Introduction

There are a number of process algebras to specify and analyze the requirements of *distributed mobile real-time systems* (DMRS). Different from other systems, DMRS requires both mobile and temporal requirements to be specified and analyzed [1]: Mobility allows various types of inter-process relations to be specified and analyzed; Temporalness does the temporal conditions for the inter-process relations to be specified and analyzed. However there were limitations in the existing process algebras to represent both mobile and temporal properties together at the same specification and analysis. This paper proposes a new process algebra, called *dT-Calculus*, which can express effectively both mobile and temporal properties of processes in DMRS, in order to specify and analyze the mobile and temporal requirements of DMRS.

Among the existing process algebras, *Timed pi-Calculus* [2] and *d-Calculus*[3] can be classified as the ones that can represent both mobile and temporal properties of DMRS. The main characteristics of the properties can be summarized as follows:

1) Timed pi-Calculus: The existing *pi-Calculus* [4] expresses process movements indirectly by using the notion of *value passing*. Timed pi-Calculus is the timed version of pi-Calculus, which allows *time-stamp* and *clock* be passed additionally during value passing: the termporal requirements of the process movements can be specified.
2) d-Calculus: This is a process algebra that can express direct process movements into or out of other processes by using the 4 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.

The above algebras are designed to specify the mobile and temporal requirements of process movements, but there are some limitations to specify the requirements fully, considering different temporal properties of the requirements:

1) Timed pi-Calculus: It allows various types of temporal requirements to be specified, but the actual execution time of action itself and type of

movements are not possible to specify in the requirements.

2) d-Calculus: It allows various types of temporal requirements to be specified, but only simple type of temporal requirements for process movements is 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 these limitations, this paper proposes a process algebra, namely, *dT-Calculus*, which is the timed version of d-Calculus, extended for more specific temporal specification and analysis of the requirements. dT-Calculus allows the temporal properties of the actions of the DMRS processes to be expressed as follows:

- *Ready time*: The time needed before execution of action.
- *Timeout*: The maximum waiting time up to the actual execution of action is possible, after the execution of the action is ready with the 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*: Periodic repetition of an action or a process.

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

This paper is organized as follows. Section 2 introduces some of the existing process algebras with temporal properties. Section 3 introduces the basic algebra for dT-Calculus, that is, d-Calculus. Section 4 describes syntax and semantics of dT-Calculus, focusing on its temporal properties. Section 5 demonstrates usability of dT-Calculus with a simple example. Section 6 shows some comparison dT-Calculus with other process algebras. Section 7 introduces a tool, called SAVE, which is developed, on the 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

One of the best known process algebra to specify and analyze the temporal property is *Timed pi-Calculus*. It is an timed version of pi-Calculus designed of Milner [4], adding the temporal property for process movements. Fig. 1 shows the syntax of Timed pi-Calculus.

In the *send* and *receive* actions, 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:

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

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,

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

It specifies two temporal conditions with clock: $(e > 5)$ represents a condition for a local clock *e*, and $(d - t_z \le 3)$ represents a t emporal condition related to a r eceiving message. d and t_z are times on the clock for the receiving message and its time-stamp, but, since the clock ticks continuously, $(d - t_z \le 3)$ 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,

$$\overline{y}x.P' \mid y(z).Q' \mid R \xrightarrow{\tau} P' \mid Q'\{x/z\} \mid R$$

As shown in Fig. 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*.

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

Fig. 1 Syntax of Timed pi-Calculus

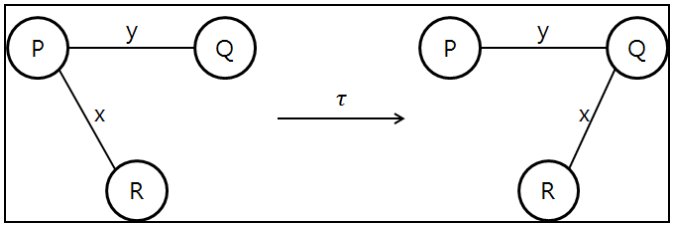


Fig. 2 Movement in Timed pi-Calculus

3 Previous 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 and all of them are synchronously defined.

3.1 Syntax

The syntax of d-Calculus is shown in Fig. 3 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.
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 its 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 nondeterministically 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 the action A.
9) Empty: No action.
10) Send/Receive: Communication between processes, exchaning a message by a channel r. t represents the 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 terminatee.
15) Exit process: Termination of its own process. All internal processes will be terminated at the same time.

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

Fig. 3 Syntax of d-Calculus

Generally all the movements are synchronous. In order for a process to move in or out of another process, the moving process (*mover*) needs a permission from the target process. Reversely, in order for a process to be moved in or out of another process forcefully, the moving process needs a 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 DMRS 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 permission process. It implies that the movement cannot be allowed without the 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 Fig. 4.

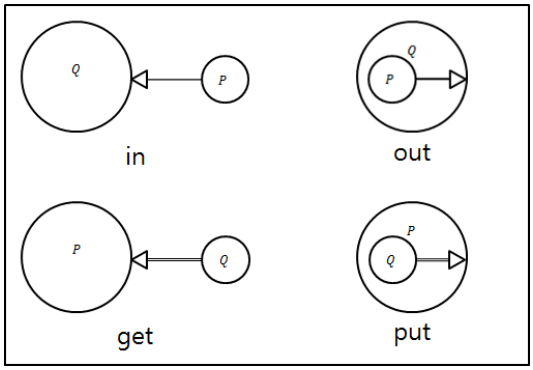


Fig. 4 Pictorial View of d-Calculus Movements

4 dT-Calculus

dT-Calculus is the process algebra developed to specify and analyze the process movements of DMRS 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 the different types of temporal properties for period and sporadic actions or processes, with the additional syntax and semantics accordingly.

4.1 Time Properties

There are five temporal properties in dT-Calculus: *ready time*, *timeout*, *execution time*, *deadline*, and *period*. The first 4 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 properties are as follows:

1) Ready time: It represents the waiting time for an action. At the point of the action in a process, the process has to wait for 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 to *timeout* and *deadline*.

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. Fig. 5 shows a pictorial representation of the relations among the temporal properties.

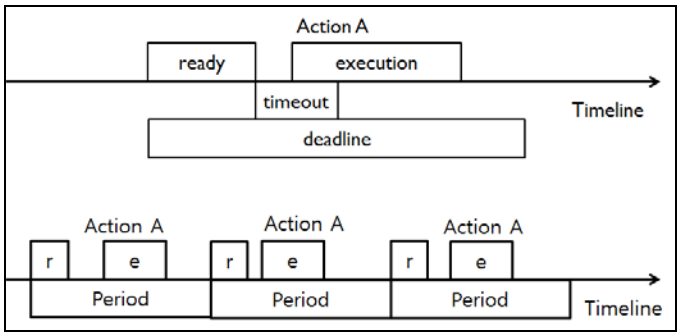


Fig. 5 Relations among Temporal Properties

4.2 Syntax

The syntax of dT-Calculus is shown in Fig. 6, and is defined as follows:

1) Action: Actions performed by a process.
2) 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 a periodic action or process: *p* for period and *n* for the number of repetition.
3) Timed process: Process with temporal properties.
4) 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.
5) 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 its external without the permission of the external.
6) Channel: *A* channel *r* of *P* to communicate with other processes. *t* implies the time needed for the communication through the channel.
7) Choice: Only one of *P* and *Q* will be selected non-deterministically for execution.
8) Parallel: Both *P* and *Q* are running concurrently.
9) Exception: *P* will be executed. But *F* will be executed in case that *P* is out of timeout or deadline.
10) Sequence: *P* follows after action *A*.
11) Empty: No action.
12) Send/Receive: Communication between processes, exchanging a message by a channel *r*.
13) Request: Requests for movement. *p* and *k* represent *priority* and *key*, respectively.
14) Permission: Permissions for movement.
15) Create process: Creation of a n ew internal process. The new process cannot have a higher priority than its creator.
16) Kill process: Termination of other processes. The terminator should have the higher priority than that of the terminatee.
17) Exit process: Termination of its own process. All internal processes will be terminated at the same time.

The biggest difference of dT-Calculus with d-Calculus is the notion of *timed* action and processes. In d-Calculus, the temporal property is simplely defined with *t* 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 it, 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
$\mid A^{p,n}_{[r,to,e,d]}$	timed action
$\mid P^{p,n}_{[r,to,e,d]}$	timed process
$\mid P_{(n)}$	priority
$\mid P[Q]$	nesting
$\mid P\langle r \rangle$	channel
$\mid P + Q$	choice
$\mid P \parallel Q$	parallel
$\mid P \backslash F$	exception
$\mid A \cdot P$	sequence
$A ::= \emptyset$	empty
$\mid r(\overline{msg})$	send
$\mid r(msg)$	receive
$\mid M$	movement
$\mid C$	control
$M ::= m^p(k)\ P$	request
$\mid P\ m(k)$	permission
$m ::= in \mid out \mid get \mid put$	movement types
$C ::= new\ P$	create process
$\mid kill\ P$	kill process
$\mid exit$	exit process

Fig. 6 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 show in Table 1.

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

Table 1 Temporal Semantics of dT-Calculus

Each rules in the table are 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 execetued 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*; the values of *e* and *d* decrease accordingly as time passes.
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 accordinlgy.
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 accordinlgy.
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.

$P + P = P$	Choice(1)
$P + Q = Q + P$	Choice(2)
$(P + Q) + R = P + (Q + R)$	Choice(3)
$P \parallel \varnothing = P$	Parallel(1)
$P \parallel Q = Q \parallel P$	Parallel(2)
$(P \parallel Q) \parallel R = P \parallel (Q \parallel R)$	Parallel(3)
$P[\varnothing] = P$	Nesting(1)
$R[P] + R[Q] = R[P + Q]$	Nesting(2)
$P \parallel (Q + R) = (P \parallel Q) + (P \parallel R)$	Distributive(1)
$(a_1 + a_2) \bullet P = a_1 \bullet P + a_2 \bullet P$	Distributive(2)
$P_{[r,to,e,d]} = P_{[-,-,-,d]}$	Timed Process
$A = A_{[0,-,1,-]}$	Non-time Action
$\varnothing_{[r,to,e,d]} = \varnothing_{[-,-,e,-]}$	Empty

Table 2 Laws of dT-Calculus

4.4 Laws

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

1) Choice: Identity, commutativity and associativity laws for the Choice operation.
2) Parallel: Identity, commutativity and associativity laws for the Parallel operation.
3) Nesting: An inclusion relation for the Empty action, and a Choice operation relation for included processes.
4) Distributive: *Distributive*(1) for the distributive law of Parallel over Choice; *Distributive*(2) for the distributive law of Choice over Sequence. Note that their opposite cases are not allowed. That is, Choice over Parallel; Sequence over Choice.
5) Timed Process: Only applicable temporal property for a process is deadline.
6) Non-time Action: The action with no temporal properties is same as the one with the temporal properties of [0,-,1,-].
7) 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 specification of 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. 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:

$$A \cdot \varnothing_{[-,-,e,-]} \cdot A \cdot \varnothing_{[-,-,e,-]} \cdot A \cdot \varnothing_{[-,-,e,-]} \cdot \ldots$$

It was intended to specify the above periodic actions with empty actions, but the empty actions with fixed execution time were 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 after the immediate execution of the action.

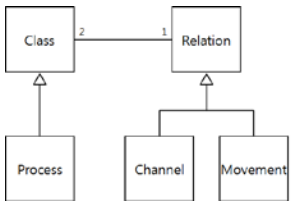
Meta-Model	Icon	
	Process	
	Channel	
	Movement	

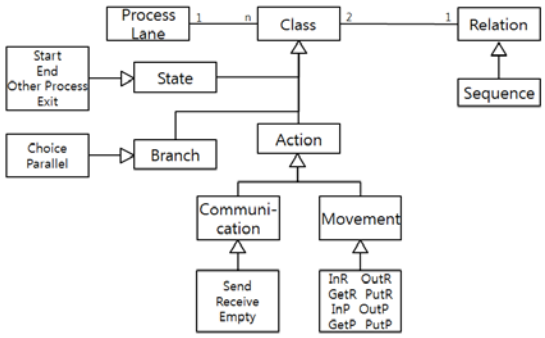
Table 3 Graphical Representation of ITL

4.6 Graphical representation

dT-Calculus specification for systems can be represented by both text and graph. Generally the graphical representation has an advantage of better understanding and comprehension of the systems over the textual representation. In dT-Calculus, there are two types of the graphical representation as follows:

1) ITL (In-the-Large): A system view showing its processes, inclusion relations and channels among the processes, and their interactions in time. The notation for the ITL view is defined in Table 3.
2) ITS (In-the-Small): A process view showing its actions and their dependencies of sequences, choices, etc. The notation for the ITS view is defined in Table 4.

Meta-Model



Icon

Process Lane		Start		End		Other Process	
Exit		Choice		Parallel		Send	
Receive		Empty		In R		Out R	
Get R		Put R		In P		Out P	
Get P		Put P		Sequence			

Table 4 Graphical Representation of ITS

5 Example

This section describes the specification of analysis of DMRS in dT-Calculus with an *Emergency Medical Systems* (EMS) example.

EMS is the system where, in case of traffic and automotive accidents, the drivers of the automotive or patients from the accidents are transported to proper medical institutes under control of the 911 rescue systems contacted by the driver or patients, or some smart devices of them.

5.1 Specification

As shown in Fig. 7 in dT-Calculus, the EMS in the example operates as follows:

1) At the time of a car accident, the driver of the car acknowledges the accident.
2) The driver attemps to move out of the car.
3) If the driver does, s/he calls 911 and informs of the accident. If not, the smart device connected with the smart car acknowledges the situation and sends an emergency signal to 911 of the accident.
4) At the call, either from the driver or the device, 911 selects appropriate ambulance from 911 for proper treatment.
5) Once the ambulance arrives at the scene of the accident, the medic takes the driver or patient on the ambulance.
6) The ambulance performs the first treatment on the driver or patient and transports him/her to the near hospital.
7) After arriving at the hospital, the ambulance takes the driver or patient off at the hospital and informs 911 of accomplishment of the transportation.
8) The driver or patient is being treated in the hospital.

$$
\begin{aligned}
EMS ::= {} & Car[Drv] \mid 911\left[Amb1_{[-,-,-,40]}\right] \mid \\
& Amb2_{[-,-,-,40]} \mid Hsp; \\
Car ::= {} & CD\left(\overline{crash}\right) \cdot Drv\ out_{[0,5,1,7]} \backslash \\
& (CE(\overline{call}) \cdot AC(open) \cdot put\ Drv); \\
Drv ::= {} & CD(crash) \cdot \\
& \left(out\ Car \cdot DE\left(\overline{call}\right) + \emptyset_{[-,-,10,-]} \cdot Car\ put\right) \cdot \\
& ((Amb1\ get \cdot AD(em) \cdot Amb1\ put) + \\
& (Amb2\ get \cdot AD(em) \cdot Amb2\ put)) \cdot HD(tr); \\
911 ::= {} & (DE(call)_{[0,5,1,7]} \cdot ((EA1(\overline{Lv1}) \cdot Amb1\ out) + \\
& EA2(\overline{Lv1}))) \backslash (CE(call) \cdot ((EA(\overline{Lv2}) \cdot amb1\ out) + \\
& EA2(\overline{Lv2}))) \cdot EA(done); \\
Amb1 ::= {} & \left(EA1(Lv1)_{[0,5,1,7]} \cdot (out\ 911)_{[0,-,5,-]}\right) \backslash \\
& (EA1(Lv2) \cdot (out\ 911)_{[0,-,5,-]} \cdot AC(\overline{open})) \cdot \\
& get\ Drv \cdot AD(\overline{em}) \cdot (in\ Hsp)_{[0,-,10,-]} \cdot \\
& put\ Drv \cdot EA\left(\overline{done}\right); \\
Amb2 ::= {} & (EA2(Lv1)_{[0,5,1,7]} \cdot \emptyset_{[-,-,3,-]}) \backslash \\
& (EA2(Lv2) \cdot \emptyset_{[-,-,3,-]} \cdot AC(\overline{open})_{[0,-,7,-]}) \cdot \\
& get\ Drv \cdot AD(\overline{em}) \cdot (in\ Hsp)_{[0,-,10,-]} \cdot \\
& put\ Drv \cdot EA\left(\overline{done}\right); \\
Hsp ::= {} & (Amb1\ in + Amb2\ in) \cdot HD(\overline{tr});
\end{aligned}
$$

Fig. 7 EMS Example in dT-Calculus

Fig. 7 shows the EMS example in dT-Calculus, specified based on the above operations. In the specified, the following action has been declared in the *Car* process to detect the case of the driver not moving out of the car autonomously:

$$Car ::= ...Drv\ out_{[0,5,1,7]} \backslash (CE(\overline{call})...$$

It implies that, if the action is not completed in the 5 time units of [0,5,1,7], the car is to call 911 automatically by means of exceptional handling of the case. In the specification of Fig. 6, the call from the car, $CE(\overline{call})$, always occurs after the call from the driver, $DE(\overline{call})$, by the operations. Therefore, the calls from the car and driver are designed to handle separately in the *Amb1* and *Amb2* processes by setting the timeout for receiving the calls in the process.

Amb1 in 911 and Amb2 out of 911 have different characteristics. Amb2 arrives at the scene earlier, it takes more time to open the doors of the car. Therefore, depending on the status of the patient, the transportation times by Amb1 and Amb2 are different.

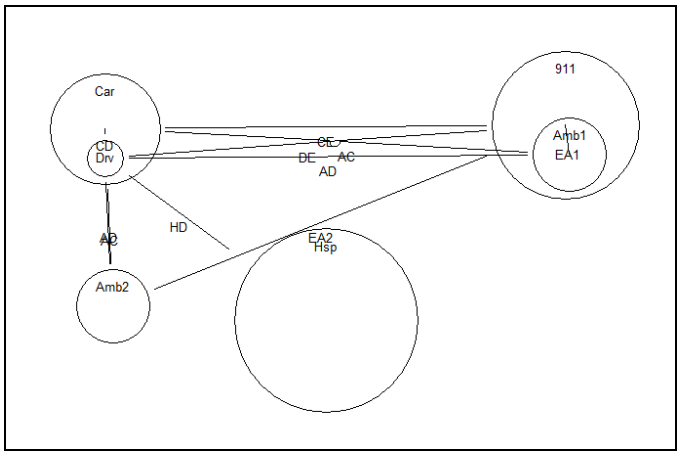


Fig. 8 ITL View of EMS System

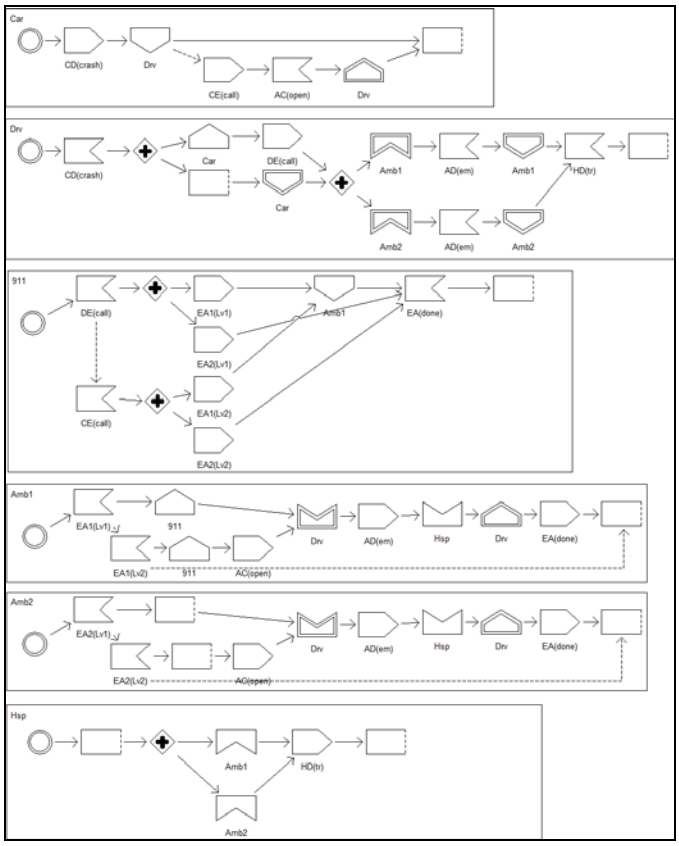


Fig. 9 ITS View of EMS System

5.2 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. Fig. 8 and 9 show the ITL and ITS views of the EMS example, respectively.

5.3 Analysis for Specification

Before execution or simulation of the EMS example, it is possible to calculate the expected minimum execution times of Amb1 and Amb2 in order to find some errors from their specification. If the expected minimum execution time of Amb1 or Amb2 is larger than its deadline, it can be obviously known, before its actual execution or simulation, that the violation of its deadline will be occurred.

There are two possible cases in the example: 1) The case that the driver is able to move out of his/her car by himself/herself; 2) The case that the driver is not able to move out of his/her car by himself/herself.

The expected minimum execution times for Amb1 can be obtained as follows:

1) Case 1: The execution time for all the actions except the first communication with 911 is 1 unit time, the expected minimum execution time will be calculated to be 20 time units from 1+5+1+1+10+1+1.
2) Case 2: In order to perform the execution of Amb1 for Case 2, a timeout or deadline error must be occurred on the $EAl(Lvl)_{[0,5,1,7]}$ action. However the sum of *ready time*, *timeout* and *execution time* is less than its *deadline*. Therefore its deadline error does not occur. Consequently the expected minimum execution time on its timeout error will be 26 time units from 5+1+5+1+1+1+10+1+1.

The expected minimum execution times for Amb2 can be obtained similarly by the same manner as follows:

1) Case 1: The execution time for all the actions except the communication with 911 is 1 un it time, the expected minimum execution time will be calculated to be 18 time units from 1+3+1+1+10+1+1.
2) Case 2: In order to perform the execution of Amb1 for Case 2, a timeout or deadline error must be occurred on the $EAl(Lvl)_{[0,5,1,7]}$ action. However the sum of *ready time*, *timeout* and *execution time* is less than its *deadline*. Therefore its deadline error does not occur. Consequently the expected minimum execution time on its timeout error will be 30 time units from 5+1+3+7+1+1+10+1+1.

From the analysis of the times, it is possible to argue that there is no error which can be found in the specification be execution, since both the expected minimum execution times for both Amb1 and Amb2 are less than 40. In addition, it is possible to predict that Amb2 consumes less time that Amb2 for Case 1, and that Amb1 consumes less time that Amb1 for Case 2.

In real execution or simulation, it is possible for some delays to occur to some actions due to synchronization with their partners. Therefore it is necessary to analyze the results of the real execution or simulation for the EMS example in order to get the real execution times.

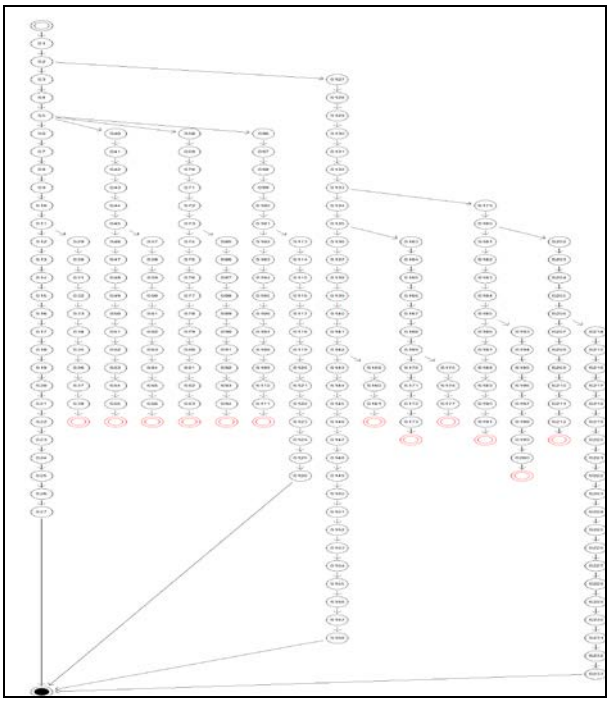


Fig. 10 Execution path of EMS System

5.4 Execution

As shown in Fig. 10, there are total 16 paths: 4 normal execution paths and 12 abnormal paths. The abnormal ones are due to improper selection of the branches at the *choice* statements. For example, it is the case that 911 sent a signal to Amb1 and moved to the scene, but the patient waits for Amb2.

For the normal execution paths, it is possible to perform analysis of their temporal properties.

Firstly, the case that the patient moves out of the car safely first is as follows:

1) T1: An accident occurs and the driver moves out of the car safely by himself or herself.
2) T2: The driver moves out of the car.
3) T3: The driver calls 911.
4) T4: 911 calls its ambulance to go to the place where the accident occurred.

Here the time to arrive at the scene by the ambulance is different, depending on which ambulance is chosen to handle the situation by 911. Therefore, the time that the ambulance arrives at the scene is defined as *t*:

5) T4+*t*: The ambulance moves out of 911 and arrives at the place.
6) T5+*t*: The driver hops on the ambulance.
7) T6+*t*: The medic in Ambulance performs the first aid treatment for the driver.
8) T16+*t*: The ambulance arrives at a hospital.
9) T17+*t*: The medic helps the driver to move into the hospital.
10) T18+t: A doctor in the hospital treats the driver and the ambulance sends 911 a message of completing its mission.

Amb1 consumes total 23 unit times, since *t*=5; Amb2 does total 21 unit time, since *t*=3.

Secondly, the case that the patient does not move out of the car safely is as follows:

3) T1: An accident occurs and the driver is unconscious.
4) T6: The car acknowledges that the driver cannot move out of the car after the accident.
5) T7: The car sends 911 a message of the accident.
6) T8: 911 calls its ambulance to go to the place where the accident occurred.

Similar to the first case, here the time to arrive at the scene by the ambulance is different, depending on which ambulance is chosen to handle the situation by 911. The time that the ambulance arrives at the scene is defined as *t*:

7) T8+*t*: The ambulance moves out of 911 and arrives at the place.

Here the time to open the doors of the car is difference. Therefore the time is defined as *d*.

8) T8+*t*+*d*: The medic opens the driver-side door of the car.
9) T9+*t*+*d*: The medic moves the driver out of the car.
10) T10+*t*+*d*: The medic moves the driver into the ambulance.
11) T11+*t*+*d*: The medic performs the first aid treatment for the driver.
12) T21+*t*+*d*: The ambulance arrives at a hospital.

13) T22+*t*+*d*: The medic moves the driver into the hospital.
14) T23+*t*+*d*: A doctor in the hospital treats the driver and the ambulance sends 911 a message of completing its mission.

For Amb1, *t*=5, *d*=1; for Amb2 *t*=3, *d*=7. Therefore Amb1 consumes 29 unit times; Amb2 consumes 33 unit times. The results of the execution are summarized in Table 5.

	Case 1: Driver moves out of the car	Case 2: Driver is unconscious
Amb1	23 time unit	29 time unit
Amb2	21 time unit	33 time unit

Table 5 Results of the Example

As the results of the execution, it is known that the deadlines for the Amb1 and Amb2 processes are satisfied. In addition, it is possible to determine which ambulance to send for treatment depending on the status of the patient. In case that the patient moves out of the car autonomously, Amb2 is better since it arrives earlier than Amb1. However in case that the patient is not able to do so, Amb1 is better since it opens the doors faster even though it arrives little late.

6 Comparative Analysis

Timed pi-Calculus is a process algebra that is designed to specify and analyze smart 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.

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 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 limited 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; 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 possible.

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 processes 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.

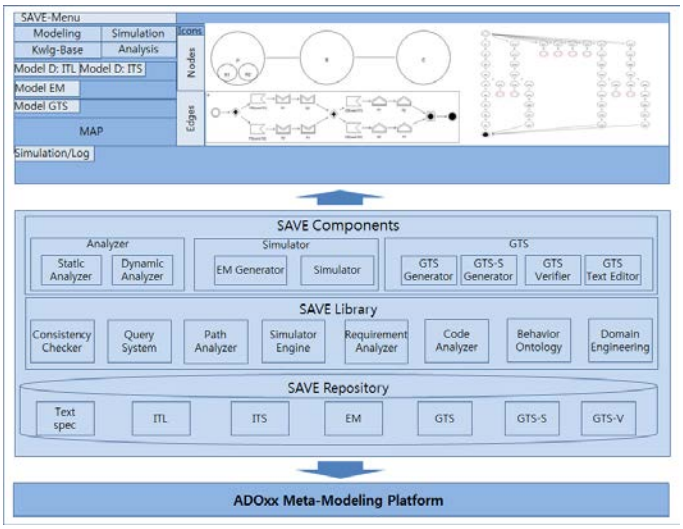


Fig.11 Architecture of SAVE

7 SAVE

In order to demonstrate the feasibility of the approach in the paper, a tool, called SAVE *(Specification, Analysis, Verification and Evaluation*), has been developed on the ADOxx Meta-Modeling Platform. There are three basic models defined in SAVE: *Specification*, *Execution* and *Simulation*. Fig. 11 shows the architecture of SAVE.

The first step to use SAVE for analysis is to specify systems in dT-Calculus. There are two specification models in SAVE, as shown in Fig. 12 and 13 for the

EMS example: ITL (In-The-Large) and ITS (In-The-Small).

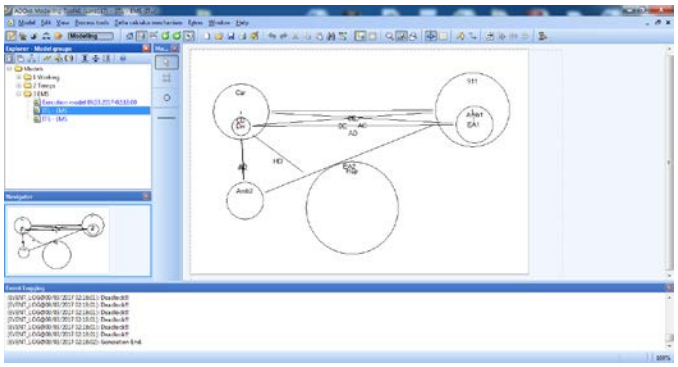


Fig. 12 ITL Model in SAVE

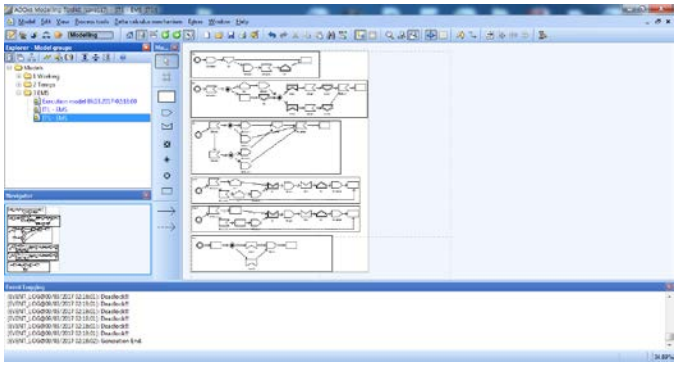


Fig. 13 ITS Model in SAVE

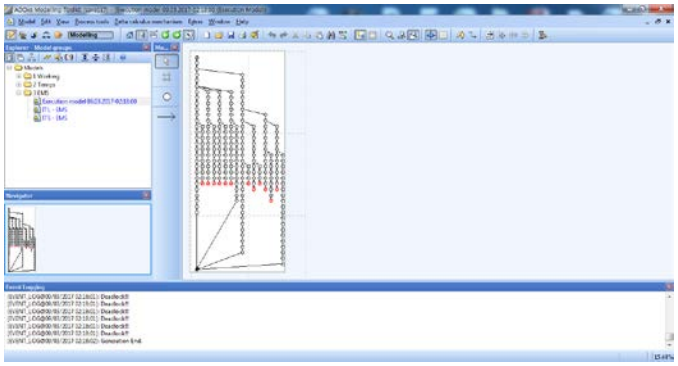


Fig. 5 Execution Model in SAVE

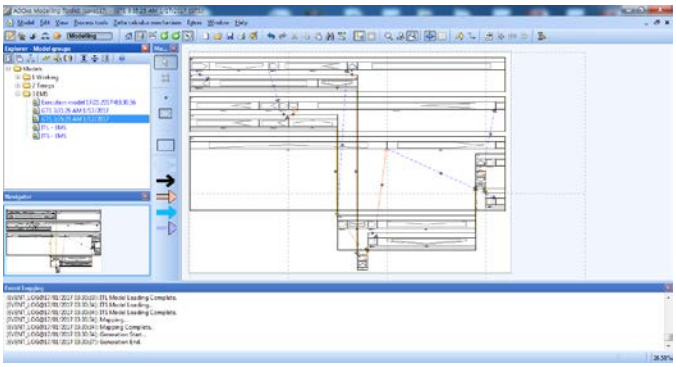


Fig. 6 Simulation Model in SAVE

From specification, the execution model can be automatically generated by the execution model generator. The execution model reveals all possible execution paths and determines whether each path is of normal or deadlock. Fig. 14 shows the execution model for the EMS example.

After generating an execution model, the simulation model for each execution path is automatically generated. The simulation model is represented in GTS (*Geo-Temporal Space*), where all the execution and movements resulted in the path are described in the model in detail. Based on the simulation model, it is possible to analyze and verify the temporal requirements of DMRS, using GTS Logic [7]. Fig. 15 shows the simulation model for the first path of the EMS example in Fig. 14.

8 Conclusion and Future research

This paper proposed a new process algebra, called dT-Calculus, for mobile and temporal specification and analysis of Distributed Mobile Real-time Systems. The algebra extended d-Calculus for specification and analysis of a v ariety 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.

Presently in the paper, the process algebra for specification and analysis is proposed. In the future research, the different types of verification methods are developed to demonstrate the usability of dT-Calculus, including SAVE with a verification model.

References:

[1] Haxthausen AE, Peleska Jan. Formal development and verification of a distributed railway control system. Software Engineering IEEE Transactions on Vol. 26, No. 8. August. 2000. pp. 68 7-701.
[2] N.Saeedloei and G.Gupta, Timed π-Calculus, Trustworthy Global Computing. TGC 2013. L ecture Notes in Computer Science, vol 8358. Springer, Cham, 2014.
[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] R. Milner, J. Parrow and D. Walker, "A calculus of mobile processes(i-ii)," Information and Computation, pp. 1-77, 1992.
[5] 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.
[6] 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.
[7] S. Lee, Y. Choe and M. Lee, A Dual Method to Model IoT Systems, International Journal of Mathematical Models and Methods in Applied Sciences, Volume 10, pp. 210-219, 2016.

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 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).