δ-CALCULUS: PROCESS ALGEBRA TO MODEL SECURE MOVEMENTS OF DISTRIBUTED MOBILE PROCESSES IN REAL-TIME BUSINESS APPLICATIONS

Complete Research

Yeongbok Choe, Chonbuk National University, Jeonju, Republic of Korea, cyb0612@gmail.com

Moonkun Lee, Chonbuk National University, Jeonju, Republic of Korea, moonkun@jbnu.ac.kr

Abstract

Some process algebras were applied to enterprise business modelling for formal specification and verification. However the algebras mostly dealt with stationary cases but the distributed and mobile. π-calculus and mobile ambient can be considered for the distributed and mobile, especially to represent the movements of distributed real-time business processes. However there are some limitations to model the movements: 1) π-calculus passes the name of port for indirect movements, and 2) mobile ambient uses ambient to synchronize asynchronous movements forcefully. As a solution to the limitations, this paper presents new process algebra, called δ-calculus, to specify direct and synchronous movements of business processes over geo-temporal business space, where a process can be nested in another process, and whose configuration is changed by the movements. Any violation of safety or security of the systems caused by the movements can be detected and prevented by the properties of the movements: synchrony, priority and deadline. It means that any movement must get the proper permission with the required priority in time. In the enterprise modelling, it will be critical to adapt this kind of concept for business safety and security. A tool, called SAVE, was developed on ADOxx meta-modelling platform to demonstrate the concept.

Keywords: δ-Calculus, Process Algebra, Movements, Autonomy, Synchrony, Time, Space, Security

1 Introduction

Process algebras, such as CCS (R.Milner, 1989), were applied to the enterprise business modelling for formal specification and verification (R. Singer and M. Teller, 2012). However the algebras mostly dealt with stationary business applications, not for distributed mobile real-time business applications. This paper presents new process algebra, namely, *δ-calculus*, to apply a process-algebraic method to model the distributed mobile real-time business applications.

There are a number of process algebras that can be considered to specify distributed mobile processes in real-time business applications. Among these, *π-calculus* (Milner, R., J. Parrow and D. Walker, 1992) and *mobile ambient* (Cardelli, L., Gordon, A., 1998) can be the best candidates since they are capable of specifying the movements of the business processes in the applications. However there are the following limitations to specify the movements:

1) π-calculus: Robin Milner extended the basic notion of CCS (R.Milner, 1982) to model the movement of processes based on the notion of value passing. Instead of representing the actual movements of a process (actor) from one process (source) to another process (target), the name of a port of the actor is passed from the source to the target to represent the movement of the actor. It can be considered as the imaginary indirect representation of the actual direct movements.

2) Mobile ambient: Luca Cardelli and Andrew D. Gordon designed this algebra with the new notion of ambient, which assists the movements of processes synchronously. Ambient provides a means of controlling asynchronous movements in synchronous manner, but it increases the complexity of specification with the additional dimension of the ambient. Consequently the specification and analysis of the movements become very complex and complicated. Further some processes can fall into some deadlock state since no movement is possible for nested ambient.

In order to solve these limitations, this paper introduces new process algebra, namely, *δ-calculus*, with the following distinctive properties of the movements. Note that a process can be nested in another process under a conceptual geographical space.

1) Autonomy: There are two types of autonomy-related movements: active and passive. The active movements are the autonomous movements of a process into or out of other process. The passive movements are the heteronomous movements of a process caused by other process.

2) Synchrony: There are two types of movements: synchronous and asynchronous. The movements in the calculus are basically synchronous, which means that there is a co-movement for every active and passive movement. Each asynchronous movement is accomplished by its dedicated synchronous co-movement which waits indefinitely for its conditional passive or active movement.

3) Priority: A discrete level of priority can be assigned to each process. Therefore it is possible to control the accessibility of a process to another process. This means that it can guarantee the safety and security of the systems at any time of execution of the movements.

4) Time: The temporal properties of movements can be specified, including *ready time*, *execution time*, and *deadline*. In addition, the proper actions for the violations of deadlines can be specified, including exceptions.

It can be seen that these properties provide the capability to overcome the stated limitations of the process algebras properly. Further it can be seen that one of the best application areas for the calculus would be enterprise business process modelling for business safety and security over distributed mobile real-time environments.

The organization of the paper is as follows. Section 2 introduces the basic notions of δ-calculus, including the syntax, semantics, laws and the graphical representation of the calculus. Section 3 describes the basic properties of the calculus: movement actions. Section 4 demonstrates the applicability

of the calculus with a security example. Section 5 presents the analysis of the calculus with *π-calculus* and *mobile ambient* on the properties. Section 6 describes a tool, called SAVE, for the calculus, which has been developed on ADOxx meta-modeling platform (Fill, H. and Karagiannis, D., 2013). Finally conclusion and future researches are made in Section 7.

2 δ-calculus

2.1 Overview

δ-calculus is a process algebra to specify the movements of business processes in distributed mobile real-time environments with the special notion of movements over a conceptual geographical space.

The movements in δ−Calculus are basically synchronous. Therefore it can prevent the problems caused by asynchronous movements. Further asynchronous movements are possible with restrict requirements to control the movements in the synchronous manner.

$P ::= nil$	// Inaction	$M ::= m_t^p(k)\ P$	// Movement Request
$\mid P_{(n)}$	// Priority	$\mid \bar{m}_t^p(k)\ P$	// Hide Movement
$\mid P[Q]$	// Nesting		// Request
$\mid P[\bar{Q}]$	// Hide	$\mid P\ m(k)$	// Movement Permission
$\mid P\langle r\rangle$	// Port	$\mid P\ \bar{m}(k)$	// Hide Movement
$\mid P\backslash F$	// Restriction		// Permission
$\mid P + Q$	// Choice		
$\mid P \parallel Q$	// Parallel	$m ::= in \mid out \mid get \mid put$	// Movement Action
$\mid r(a).P$	// Communication		
$\mid P \triangle_t (S, T, I)$	// Communication Scope	$C ::= new\ P$	// Create Process
$\mid P\square_{[l,u]}(S, L, U, K, R, I)$	// Movement Scope	$\mid kill\ P$	// Kill Process
$\mid M.P$	// Movement	$\mid exit$	// Exit Process
$\mid C.P$	// Process Control		

*Figure 1. The Syntax of **δ**-Calculus*

2.2 Syntax

The syntax of δ-calculus is shown in Figure 1 with the following definitions:

1) Inaction (0): No action for a process.
2) Priority ($P_{(n)}$): It represents the priority of the process *P* with a natural number $n \geq 0$. The higher number represents the higher priority. By default, the priority 0 is defined to be the highest priority for convenience.
3) Nesting ($P[Q]$): *P* contains *Q*. The inner process (*Q*) is controlled by the outer process (*P*). Therefore it is not possible for the inner process to communicate with other process outside without permission of the outer process. Even though the inner process has the higher priority than that of the outer process, there is no change in the permission policy. However the higher priority allows the inner process to move out of the outer process without any permission
4) Hide ($P[\bar{Q}]$): *Q* is hiding in *P* and cannot be detected outside of *P*. The overline on *Q* indicates the *hide* state of *Q*. In the hide state, the hidden process *Q* can share the same priority and ports of *P*, undetectably by *P* or any process outside of *P*.
5) Port ($P\langle r\rangle$): It represents a port of *P*, that is, *r*, through which *P* can communicate with others.

6) Restriction (P\F): It restricts actions or movements of Process *F* in *P*.

7) Choice ($P + Q$): Only one of *P* and Q will be selected nondeterministically for execution if their priorities are same. If the priorities are different, the process with the higher will be selected.

8) Parallel ($P_1 \parallel P_2$): Both P_1 and P_2 are running concurrently.

9) Communication ($r(a).P$): *P* communicates with other process connected with the port *r* to pass the message *a*. The mode of passing is indicated by $\bar{a}$ for sending and a for receiving.

10) Communication scope ($P \triangle_t (S, T, I)$): It represents all the execution branches of *P* by the deadline indicated by $0 \leq t \leq \infty$ for communication. *S* follows after the normal termination of *P*, *T* at the violation of *t*, and *I* at the interrupt from outside.

11) Movement Scope ($P\square_{[l,u]}(S, L, U, K, R, I)$): It represents all the execution branches of *P* by the deadline indicated by $0 \leq l < u \leq \infty$ for movements. *S* follows after the normal termination of *P*, *L* at the violation of the lower time bound, *l*, *U* at the violation of the upper, *u*, *K* at the key violation, such as password, *R* at the priority violation, and *I* at the interrupt from outside.

12) Movement request ($m_t^p(k)\ P$; $\bar{m}_t^p(k)\ P$): It represents a request for a movement to or from a target process with a key. There are two types of such movements: one for the normal movement (*m*) and another for the *hide* movement ($\bar{m}$). Here *t*, *p* and *k* represent the time, priority, and the password for the movement, respectively.

13) Movement permission ($P\ m(k)$; $P\ \bar{m}(k)$): It represents a permission for the movement from the above request. Similarly to the request, there are two types of the movements: one for the normal movement (*m*) and another for the *hide* movement ($\bar{m}$).

14) Create process (new P): It represents the action to create a process outside with a restriction that a new process cannot have a higher than that of the creator.

15) Kill process (kill P): It represents the action to terminate other process whose priority should be lower than that of the terminator or non-*hide* process.

16) Exit process (exit): It represents the action to terminate itself: All of its child processes will be terminated automatically and hierarchically. It is different from *nil* operation, which can be considered as a deadlock.

Basically the movements in the calculus are defined as synchronous: The request to enter or move out of has to get a permission from the target process in the autonomous case, and, similarly, the request to make another process to enter into or move out of itself has to get a permission from the target process in the heteronomous case, too.

The main reason for the strict synchronous movements is to control all the movements of the mobile processes under the given policy of the mobility determined by the degree of permission with certain priority criteria. This will guarantee the safety and security of the target systems by preventing unnecessary and unauthorized movements from being allowed. Further the synchrony of the movements can be tightly coordinated by specifying the temporal properties in the movement scope just as defined in the communication scope.

2.3 Semantics: Transition Rules

Semantics of δ-calculus are described by transition rules as shown in Tables 1 and 2. The semantics in Table 1 are as follows:

1) Action: It is a transition rule for the execution of an communication action *r*(*a*) on a port *r* with a message *a*. It does not require any premise for the transition, after which *P* will be executed next.
2) Choice: *ChoiceL* and *ChoiceR* transition rules can be equally chosen under the same premise.

ChoiceP rule is determined by priority.

3) Parallel: The transition of a process in *ParIL* and *ParIR* rules does not influence its parallel process. But *ParCom* rule requires that two parallel processes must synchronously interact with each other in order to make their corresponding transitions.
4) Restriction: This rule allows only actions not defined in *F* to be executed by the rule.
5) Communication Scope: *CScopeC* rule represents time passing for an action of *P* in execution. *CScopeS* is for the transition to *S* at the normal termination of *P* within deadline. *CScopeT* for the transition to *T* at the abnormal termination of *P* due to deadline violation. *CScopeI* for the transition to *I* at any interrupt.
6) Hide: *HideC* rule represents that a *Hide* process *Q* at *P* can make an unauthorized transition by a port of *P*.

Action	$\frac{-}{r(a).P \xrightarrow{r(a)} P}$	ChoiceL	$\frac{P_{(n)} \xrightarrow{a} P'_{(n)}}{P_{(n)} + Q_{(n)} \xrightarrow{a} P'_{(n)}}$	ChoiceR	$\frac{Q_{(n)} \xrightarrow{a} Q'_{(n)}}{P_{(n)} + Q_{(n)} \xrightarrow{a} Q'_{(n)}}$
ParIL	$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q}$	ParIR	$\frac{Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{a} P \parallel Q'}$	Par Com	$\frac{P \xrightarrow{a} P', Q \xrightarrow{\bar{a}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$
CScopeC	$\frac{P \xrightarrow{a} P'}{P\backslash F \xrightarrow{a} P'\backslash F}(a, \bar{a} \notin F)$	CScopeS	$\frac{P \xrightarrow{a} 0}{P \vartriangle_t (S, T, I) \xrightarrow{a} S}(t > 0)$	CScopeT	$\frac{T \xrightarrow{a} T'}{P \vartriangle_t (S, T, I) \xrightarrow{a} T'}(t = 0)$
CScopeI	$\frac{T \xrightarrow{a} T'}{P \vartriangle_t (S, T, I) \xrightarrow{a} T'}(t = 0)$		HideC	$\frac{Q\langle B\rangle \xrightarrow{r(a)} Q'\langle B\rangle}{P[\overline{Q}]\langle A\rangle \xrightarrow{r(a)} P[\overline{Q'}]\langle A\rangle}(r \in A, r \notin B)$	
ChoiceP	$\frac{P_{(n)} \xrightarrow{a} P_{(n)}', Q_{(m)} \xrightarrow{b} Q_{(m)}'}{P_{(n)} + Q_{(m)} \xrightarrow{a} P_{(n)}'}(n > m)$		Restriction	$\frac{P \xrightarrow{a} P'}{P\backslash F \xrightarrow{a} P'\backslash F}(a, \bar{a} \notin F)$	

*Table 1. Communication Semantics of **δ**-Calculus*

The semantics in Table 2 are as follows:

1) Movement: *In*, *Out*, *Get*, *Put* transition rules represent the general synchronous movements. Each movement must have its corresponding co-movement with the same key. *InH*, *OutH*, *GetH*, *PutH* rules are same as the previous rules, but for the movements under the *Hide* condition. Similarly, *InP*, *OutP*, *GetP*, *PutP* rules for the movements with priority. However these movements can be asynchronous by the priority. It means that these movements with higher priority do not require the permission by the co-movements.
2) Movement Scope: *MScopeC* rule represents time passing for a movement of *P*. *MScopeS* is for the transition to *S* at the normal termination of *P* within deadline. *MScopeL* for the transition to *L* at the abnormal termination of *P* at the lower bound violation. Similarly, *MScopeU* for the transition to *U* at the abnormal termination of *P* at the upper bound violation. *MScopeK* for the transition to *K* at the key miss-match. *MScopeR* for the transition to *R* at the priority violation. *MScopeI* for the transition to *I* at any interrupt.

In	$\frac{P \xrightarrow{in_t(k)\ Q} P', Q \xrightarrow{P\ in\ (k)} Q'}{P \parallel Q \xrightarrow{\delta} Q'[P']}$	GetH	$\frac{P \xrightarrow{\overline{get}_t(k)\ Q} P', Q \xrightarrow{P\ \overline{get}\ (k)} Q'}{P \parallel Q \xrightarrow{\delta} P'[\overline{Q'}]}$
Out	$\frac{P \xrightarrow{out_t(k)\ Q} P', Q \xrightarrow{P\ out\ (k)} Q'}{Q[P] \xrightarrow{\delta} P' \parallel Q'}$	PutH	$\frac{P \xrightarrow{\overline{put}_t(k)\ Q} P', Q \xrightarrow{P\ \overline{put}\ (k)} Q'}{P[\overline{Q}] \xrightarrow{\delta} P' \parallel Q'}$
Get	$\frac{P \xrightarrow{get_t(k)\ Q} P', Q \xrightarrow{P\ get\ (k)} Q'}{P \parallel Q \xrightarrow{\delta} P'[Q']}$	InP	$\frac{P_{(n)} \xrightarrow{in_t^p(k)\ Q_{(m)}} P'_{(n)}}{P_{(n)} \parallel Q_{(m)} \xrightarrow{\delta} Q_{(m)}[P^{n\prime}]}(n \geq m)$

Put	$\frac{P \xrightarrow{put_t(k)\ Q} P',\ Q \xrightarrow{P\ put(k)} Q'}{P[Q] \xrightarrow{\delta} P' \parallel Q'}$	OutP	$\frac{P_{(n)} \xrightarrow{out_t^p(k)\ Q_{(m)}} P'_{(n)}}{Q_{(m)}[P_{(n)}] \xrightarrow{\delta} P'_{(n)} \parallel Q_{(m)}} (n \geq m)$
InH	$\frac{P \xrightarrow{\overline{in}_t(k)\ Q} P',\ Q \xrightarrow{P\ \overline{in}(k)} Q'}{P \parallel Q \xrightarrow{\delta} Q'[\overline{P'}]}$	GetP	$\frac{P_{(n)} \xrightarrow{get_t^p(k)\ Q_{(m)}} P'_{(n)}}{P_{(n)} \parallel Q_{(m)} \xrightarrow{\delta} P'_{(n)}[Q_{(m)}]} (n \geq m)$
OutH	$\frac{P \xrightarrow{\overline{out}_t(k)\ Q} P',\ Q \xrightarrow{P\ \overline{out}(k)} Q'}{Q[\overline{P}] \xrightarrow{\delta} P' \parallel Q'}$	PutP	$\frac{P_{(n)} \xrightarrow{put_t^p(k)\ Q_{(m)}} P'_{(n)}}{P_{(n)}[Q_{(m)}] \xrightarrow{\delta} P'_{(n)} \parallel Q_{(m)}} (n \geq m)$
MScoepS	$\frac{P \xrightarrow{a} 0}{P\square_{[l,u]}(S, L, U, K, R, I) \xrightarrow{a} S} (l = 0, u > 0)$	MScoepL	$\frac{P \xrightarrow{a} 0}{P\square_{[l,u]}(S, L, U, K, R, I) \xrightarrow{a} L} (l > 0)$
MScoepU	$\frac{U \xrightarrow{a} U'}{P\square_{[l,u]}(S, L, U, K, R, I) \xrightarrow{a} U'} (u = 0)$	MScopeK	$\frac{P \xrightarrow{m_t(a)\ Q} P',\ Q \xrightarrow{P\ m(b)} Q'}{P\square_{[l,u]}(S, L, U, K, R, I) \xrightarrow{a} K} (u > 0, a \neq b)$
MScopeC	$\frac{P \xrightarrow{a} P'}{P\square_{[l,u]}(S, L, U, K, R, I) \xrightarrow{a} P'\square_{[l-t,u-t]}(S, L, U, K, R, I)}$		
MScopeR	$\frac{P_{(n)} \xrightarrow{m_t^p(a)\ Q_{(m)}} P'_{(n)}}{P_{(n)}\square_{[l,u]}(S, L, U, K, R, I) \xrightarrow{m_t^p(a)\ Q_{(m)}} R} (u > 0, n < m)$		
MScopeI	$\frac{I \xrightarrow{a} I'}{P\square_{[l,u]}(S, L, U, K, R, I) \xrightarrow{a} I'} (u > 0)$		

*Table 2. Movement Semantics of **δ**-Calculus*

$P \equiv P$	L1. Reflexive	$(P \parallel Q) \parallel R \Longrightarrow P \parallel (Q \parallel R)$	L8. Parallel assoc
$P \equiv Q \Rightarrow Q \equiv P$	L2. Symm	$R[P] \parallel R[Q] \not\equiv R[P \parallel Q]$	L9. Separate
$P \equiv Q, Q \equiv R \Longrightarrow P \equiv R$	L3. Trans	$P \parallel 0 \equiv P$	L10. Zero par
$P \equiv Q \Longrightarrow P \parallel R \equiv Q \parallel R$	L4. Parallel	$P[0] \equiv P$	L11. Zero nesting
$P \equiv Q \Longrightarrow R[P] \equiv R[Q]$	L5. Nesting	$Q\ in.\ P \parallel in\ P.\ Q \rightarrow P[Q]$ if $P \parallel Q$	L12. Red in
$P \equiv Q \Longrightarrow A(a, n).\ P \equiv A(a, n).\ Q$	L6. Action	$Q\ out.\ P \parallel out\ P.\ Q \rightarrow P \parallel Q$ if $P[Q]$	L13. Red out
$P \parallel Q \Longrightarrow Q \parallel P$	L7. Paralle comm	$get\ Q.\ P \parallel P\ get.\ Q \rightarrow P[Q]$ if $P\|\|Q$	L14. Red get
		$put\ Q.\ P \parallel P\ put.\ Q \rightarrow P \parallel Q$ if $P[Q]$	L15. Red put
$b.\ P\square_{[l,u]}(S, L, U, K, R, I) = b.\left(P\square_{[l-t,u-t]}(S, L, U, K, R, I)\right) + E$ if $u - t > 0$ and $a \neq b$			L16. ScopeCT
$P\square_{[l,0]}(S, L, U, K, R, I) = U$	L17. ScopeTU	$a\square_{[l,u]}(S, L, U, K, R, I) = L$ if $l > 0$	L18. ScopeTL
$nil\square_{[l,u]}(S, L, U, K, R, I) = I$ if $u > 0$			L19. ScoepI

*Table 3. Laws of **δ**-calculus*

2.4 Laws

The laws of δ-calculus are listed in Table 3. These are algebraic laws used to execute transitions of processes in algebraic terms or reductions. These laws are classified in the following categories:

1) Structural laws: These laws are used just to rearrange the textual representations of processes in terms of equivalence relations as defined in L1 through L11.
2) Movement laws: These laws represent binary movement operations for *in*, *out*, *get* and *put* movements as defined in L12 through L15.
3) Scope laws: These laws represent each branch of *Scope* construct, that is, *S*, *L*, *U* , *K* and *E* as defined in L16 through L19.

These laws are used to restructure the textual configuration of processes in a system or to reduce synchronous binary interactions, that is, communication and movements, between two interactive processes as a means of execution.

2.5 Graphical Representations

δ-calculus can be represented graphically in the following two views:

1) ITL (In-The-Large) View: This is a system view with processes interacting with each other over some geographical space. In the view, a process will be represented as a node and a channel as an edge between nodes. A node can be nested in another node. Any movement can be represented as an edge, and it changes the configuration of the view. The icons for ITL view are listed in Table 4. Figure 3 in the next section shows an ITL view example.

	Process		Hide Process		Active Movement
	Passive Movement		Active Movement Hide		Passive Movement Hide
	Active Movement Priority		Passive Movement Priority		Active Movement Hide Priority
	Passive Movement Hide Priority		Communication		

Table 4. ITL Icons

2) ITS (In-The-Small) View: This is a process view with actions, including communications and movements. Actions represent the steps of execution and all the possible branches of Scope in a process. However it cannot represent the results of the movements. The icons for ITL view are listed in Table 5. Figure 4 in the section shows ITS view examples.

	Start		Other Process		End		Nil Process
	Choice		Parallel		Repeat		New
	Kill		End		Communication Scope		Movement Scope
	Send		Receive		Empty		InRequest
	InRequest Priority		InRequest Hide		InRequest Hide Priority		InPermission
	InPermission Hide		OutRequest		OutRequest Priority		OutRequest Hide
	OutRequest Hide Priority		OutPermission		OutPermission Hide		GetRequest
	GetRequest Priority		GetRequest Hide		GetRequest Hide Priority		GetPermission
	GetPermission Hide		PutRequest		PutRequest Priority		PutRequest Hide
	PuRequest Hide Priority		PutPermission		PutPermission Hide		Process Lane
	Sequence	*S*	Success	*T*	Timeout	*L*	Lower bound Timeout
U	Upper bound Timeout	*K*	Key Error	*R*	Priority Error		Interrupt

Table 5. ITS Icons

As stated, each ITL and ITS views represent a system and its processes, respectively. For the complete specification of a system, there should be an 1-1 correspondence between each node in ITL and its ITS view. Further for each interaction, there should be an 1-1 correspondence between a synchronizing action of a synchronizer and the synchronized action of its synchronizee. These correspondences guarantee the proper structural condition for syntactical completeness. If not, the proper execution of the system cannot be performed due to syntactic inconsistence.

3 Properties

3.1 Movements

In δ-calculus, synchrony is the base control method of process movements. It means that a process cannot move without permission from its synchronous partner. For example, in $P[Q[R]]$, in order for R to move out of P, R should move out of Q first. Therefore, R requires permission from P and Q in sequence.

Further δ−Calculus also allows two types of asynchronous movements as follows:

1) Asynchronous movement by parallel process: This is a method of allowing asynchronous movements for a specific process. For example, *System* is defined as follows:

$$\text{System} ::= \text{S} \parallel (\text{P in})^{\infty} \parallel (\text{P out})^{\infty}$$

$$\text{System} \parallel \text{in}_{\text{t}}\ \text{System}.\text{P} \xrightarrow{\delta} \text{System}[\text{P}]$$

Here *System* consists of two parts: one for its own task and another for permitting P to move in and out of itself, that is, $P\ in\ .\ System \ \ ||P\ out\ .\ System$. Internally, it is a set of two synchronous movements for permission, but, externally they can be seen as asynchronous movements. In this way, P appears not to require permission from *System*. In some cases, P can be defined as a group of processes or a set of anonymous processes.

2) Asynchronous movement by priority: It is possible to define an asynchronous movement with priority option. For example, System is defined as follows:

$$\text{System} ::= \text{P}_{(2)} \parallel \text{Q}_{(1)}$$

$$\text{in}_{\text{t}}^{\text{p}}\ \text{Q}_{(1)}.\text{P}_{(2)} \parallel \text{Q}_{(1)} \xrightarrow{\text{in}_{\text{t}}^{\text{p}}\ \text{Q}_{(1)}} Q_{(1)}[P_{(2)}]$$

Here the priority of P is higher than that of Q. Therefore any asynchronous movement into Q by P can be accepted by Q without any permission for its corresponding co-movement. Reversely, any asynchronous movement into P by Q will be rejected due to the lower priority.

3.2 Security

In δ-calculus, it is possible to represent a process influenced by some external environment with the characteristics of movements. A process can behave differently, depending on the conditions of whether it is contained in another process or not, or, if it is contained, of what kind of process it is contained in, etc. It implies that independently safe system can become unsafe by interactions with external processes.

4 Example

This section demonstrates applicability of δ-calculus to business process domain for security with the *CryptoLocker* example. CryptoLocker is a ransomware trojan which targeted computers running Microsoft Windows and was first observed by Dell SecureWorks in September 2013 (CryptoLocker). It

breaks into the system, infects specific files, and demands ransom for recovery from the infection. The behavior of CryptoLocker can be specified in δ-calculus as shown in Figure 2.

$$\text{System} ::= S_{(0)}\left[E_{(0)} \parallel P1_{(m)} \parallel P2_{(m)} \parallel P3_{(m)} \parallel \cdots \parallel Pn_{(m)}\right]\langle SYS, NET\rangle \parallel (F\ in)^{\infty}$$

$$F ::= P1 \parallel P2 \parallel P3 \parallel \cdots \parallel CL \parallel CP \parallel \cdots$$

$$CL ::= in_{t_1}\ S.\text{new I.new D.nil}$$

$$D ::= \text{kill CL.exit}$$

$$I ::= \overline{in}^{p}_{t_2}\ E.W$$

$$W ::= \text{nil} \triangle_{\infty} \left(\text{nil}, \text{nil}, SYS(\text{trigger}).NET(\overline{\text{download}}).W\right)$$

$$CP ::= NET(\text{download}).in_{t_3}\ S.\text{new K.}\overline{get}^{p}_{t_4}\ K.\left(get^{p}_{t_5}\ Pi.\text{new Pi}'.\text{kill Pi}\right)^{n}$$

$$.(SYS.(\overline{ReqM}).SYS(\text{sendM}))\ \triangle_{t_5}\ (OP, \text{kill K.exit}, \text{nil})$$

$$OP ::= \left(get^{p}_{t_6}\ Pi'.\text{new Pi.kill Pi}'\right)^{n}.\text{exit}$$

*Figure 2. Specification of CryptoLocker in **δ**-Calculus*

4.1 Specification

As shown in Figure 2, *System* consists of two parts: 1) *S* and its internal components and 2) the other to give permission for an external process to enter *S*. *S* contains *E*, to search files in the system, and other various processes. Other processes except *E* have the same priority of *m*. CryptoLocker operates in the following steps:

1) After *CL* enters *S*, *CL* creates *I* and *D* processes.
2) *D* terminates *CL* and terminates itself.
3) *I* enters *E* forcibly by priority-based asynchronous movement, becomes a *Hide* process of *E*, and waits in the *W* state.
4) If *Trigger* is met by a specific action of *S*, *W* downloads *CP* by a port of *S*, *NET*.
5) When *CP* is downloaded and entered in *S*, it creates a key outside of *S* and brings the key in.
6) *CP* brings other processes in it, creates the enciphered version of the processes with the key, and forcibly terminates the original version of the processes.
7) Once the enciphering is done, demand ransom through the system *S* with a deadline.
8) If the ransom is paid in time, the enciphering is terminated and all the enciphered processes are recovered.
9) If the ransom is not paid in time, *CP* deletes the key and terminates itself.

In order to specify the behavior of CryptoLocker, the movements of *CL*, the injection of *I* into *E* to become a Hide process, and the downloading and enciphering of *CP* are represented as movement actions. Further, in order for *W* in waiting state to respond to *Trigger*, the movement scope is defined for the interrupt by *Trigger*.

The main characteristics of the example are described in Section 4.4.

4.2 ITL View

An ITL view for the example can be represented as shown in Figure 3, based on the icons for the ITL graphical representation listed in Table 4. This is the view before execution of the example.

4.3 ITS View

ITS views for *CP*, *OP* and other processes are shown in Figure 4, based on the icons for the ITS graphical representation in Table 5. These are the views before execution of the example.

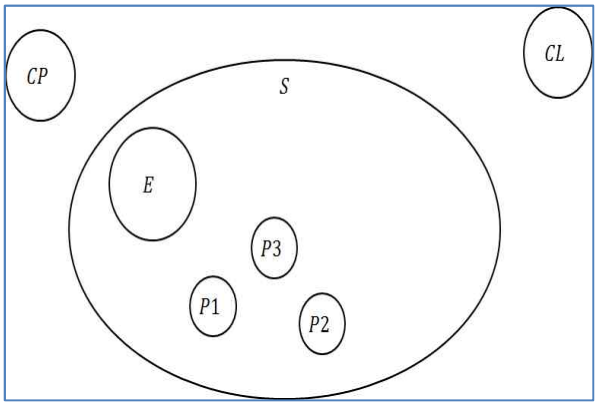

Figure 3. ITL View before Execution

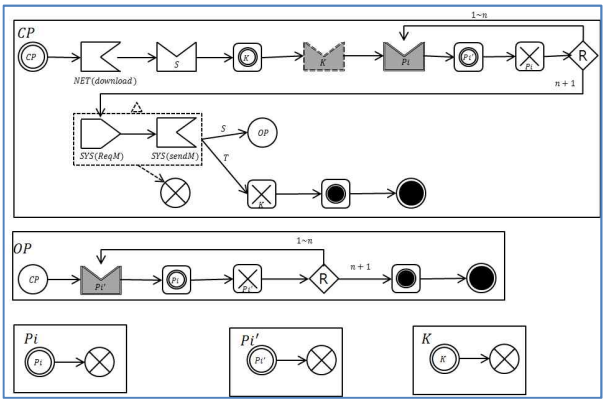

Figure 4. ITS View before Execution

4.4 Execution

The execution of the example is determined by the communication and movement actions listed in Table 6.

Synchronous Communications and Movements	Asynchronous Communications and Movements
$\mathrm{CL :: in}_{t_1}\ \mathrm{S} \sim \mathrm{S :: F\ i}$	$\mathrm{I :: \overline{in}}^{\mathrm{p}}_{t_2}\ \mathrm{E}$
$\mathrm{W :: NET}(\overline{\mathrm{download}}) \sim \mathrm{CP :: NET(download)}$	$\mathrm{CP :: \overline{get}}^{\mathrm{p}}_{t_4}\ \mathrm{K}$
$\mathrm{CP :: in}_{t_3}\ \mathrm{S} \sim \mathrm{S :: F\ in}$	$\mathrm{CP :: \overline{get}}^{\mathrm{p}}_{t_2}\ \mathrm{Pi}$
	$\mathrm{OP :: \overline{get}}^{\mathrm{p}}_{t_7}\ \mathrm{Pi'}$

Table 6. Synchronous and Asynchronous Actions

Note that there are deadlines for each communications and movements as described in subscript.

Based on these communication and movements, the example is executed in the following steps. Here *Time* indicates the virtual time for execution:

1) Time = t_1: Process *CL* enters System or Process *S*.
2) Time = $t_1 + 1$: Process *I* is created.
3) Time = $t_1 + 2$: Process *D* is created.
4) Time = $t_1 + 3$: *CL* is terminated.
5) Time = $t_1 + 4$: Process *D* is terminated.
6) Time = $t_1 + t_2 + 1$: Process *I* enters *E* and becomes a *Hide* process *W* of *E*.
7) Time = $t_1 + t_2 + 1 + a$: Process *W* gets a trigger and downloads *CP*, where a is the waiting time to receive the trigger. There is no condition for a.
8) Time = $t_1 + t_2 + 1 + a + t_3$: *CP* is downloaded and entered in *S*.
9) Time = $t_1 + t_2 + 1 + a + t_3 + 1$: Process *K* is created.
10) Time = $t_1 + t_2 + 1 + a + t_3 + 1 + t_4$: Process *K* is brought in *CP*.
11) Time=$t_1 + t_2 + 1 + a + t_3 + 1 + t_4 + (t_5 + 2) * n$: *CP* enciphers processes, P_1 through P_n.

Here are two possible choices, depending on the ransom payment. Firstly, if the ransom is paid:

12) Time = $t_1 + t_2 + 1 + a + t_3 + 1 + t_4 + (t_5 + 2) * n + b$: Process *CP* checks the payment, where *b* is the time for ransom to be paid: $b < t_6$.

13) Time $= t_1 + t_2 + 1 + a + t_3 + 1 + t_4 + (t_5 + 2) * n + b + (t_7 + 2) * n$: *CP* recovers the deciphered processes, P_1' through P_n'.

14) Time=$t_1 + t_2 + 1 + a + t_3 + 1 + t_4 + (t_5 + 2) * n + b + (t_7 + 2) * n + 1$: *CP* terminates itself.

Secondly, if the ransom is not paid:

12) Time = $t_1 + t_2 + 1 + a + t_3 + 1 + t_4 + (t_5 + 2) * n + t_6$: The deadline t_6 is over.

13) Time = $t_1 + t_2 + 1 + a + t_3 + 1 + t_4 + (t_5 + 2) * n + t_6 + 1$: *K* terminates itself.

14) Time = $t_1 + t_2 + 1 + a + t_3 + 1 + t_4 + (t_5 + 2) * n + t_6 + 2$: *CP* terminates itself.

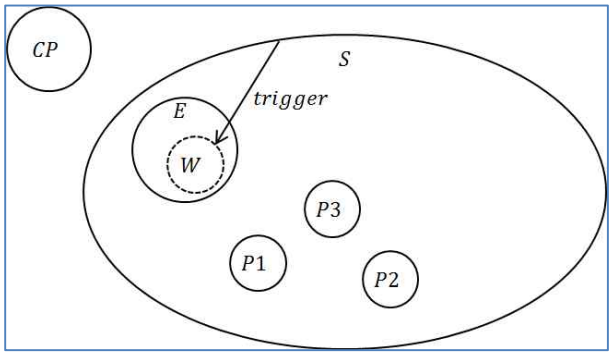

Figure 5. ITL View at Step 7.

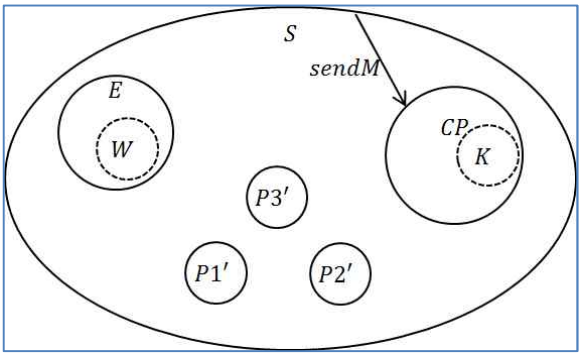

Figure 6. ITL View at Step 12.

Each movement during execution changes the configuration of the ITL view. For example, Figure 5 shows the ITL view for the example at the time of getting a trigger at the execution step 7 with Time = $t_1 + t_2 + 1 + a$. Its transition is performed by as follows:

$$\mathbf{S}[\mathrm{E}[\bar{\mathbf{W}}] \parallel \mathrm{P1} \parallel \mathrm{P2} \parallel \mathrm{P3}] \parallel \mathrm{CP} \xrightarrow{\tau} \mathrm{S}[\mathrm{E}[\bar{\mathrm{W}}] \parallel \mathrm{P1} \parallel \mathrm{P2} \parallel \mathrm{P3}] \parallel \mathrm{CP}$$

Another figure, Figure 6 shows the ITL view for the example at the time of getting the requested random at the execution step 12 with Time = $t_1 + t_2 + 1 + a + t_3 + 1 + t_4 + (t_5 + 2) * n + t_6$. Note that all processes are reciphered. Its transition is performed by as follows:

$$\mathbf{S}[\mathrm{E}[\bar{\mathrm{W}}] \parallel \mathrm{P1}' \parallel \mathrm{P2}' \parallel \mathrm{P3}' \parallel \mathbf{CP}[\bar{\mathrm{K}}]] \xrightarrow{\tau} \mathrm{S}[\mathrm{E}[\bar{\mathrm{W}}] \parallel \mathrm{P1}' \parallel \mathrm{P2}' \parallel \mathrm{P3}' \parallel \mathrm{CP}[\bar{\mathrm{K}}]]$$

4.5 Interpretation for Enterprise Business Modelling

The example shows the criteria for the safe business transaction which should be occurred in System *S*. If any process wants to interfere or interrupt the transaction, it has to move into *S* and stay there physically at the time of the transaction. In other words, *S* can be considered as a sort of *safe zone* for the business transaction. In order for S to be safe from any intrusion, such as CryptoLocker, the intruder should be out of the zone, that is, *S*, or the intruder should not be in the zone at the time of the transaction.

5 Analysis and Comparison

π-calculus is one of the best candidates to model business processes in distributed mobile real-time applications. However it is not suitable to represent the real movements of the processes, since the movements are represented by the notion of the value-passing communication. Consequently, there is severe distortion of reality for movements and, further, there are strict restrictions to express the real movements, since the movements cannot be represented directly and explicitly.

Mobile ambient is another candidate to model business processes in distributed mobile real-time applications. It introduces the notion of asynchronous movement, and assists such movements with the notion of *ambient* in order to control the unconditional asynchrony of the movement. However the inter-

actions of processes are hard to be specified and analyzed since they are strictly controlled by the underlining interactions of ambients for the processes. In other words, every interactions of the processes have to be cross-checked by the hidden underlined interactions of their ambients. Consequently, the complexity of the specification and analysis increases exponentially. Further some processes can fall into some deadlock state since no movement is allowed for nested ambient.

In comparison, δ-calculus can overcome the limitations of these algebras: It can not only express the movements of processes directly and explicitly, but also represent them in both textual and graphical manners. Therefore it is much easier to comprehend the behavior of business processes with respect to the movements. In addition, it is advantageous to differentiate the unsafe situations of the target system caused by its external environment from the safe situation of the stand-alone system, independent from the environment, especially in the business security perspective. As generally known, the security of the business application in distributed environments cannot be guaranteed completely by itself, but be supported by their appropriate external environment. In this perspective, the existing process algebras are able to specify the required characteristics of the target system itself, but there are some limitations to specify the relations between the system and its environment fully. However, in δ-calculus, such limitations can be overcome by specifying and analyzing the relations with the properties of synchrony, priority and deadline from the environment. Consequently, the calculus can be used to specify and analyze a variety of safety and security issues or protocols for the business applications in distributed mobile real-time environments. Further it can be used to detect and prevent a number of unsecure business situations caused by immigration of other processes from the external environment.

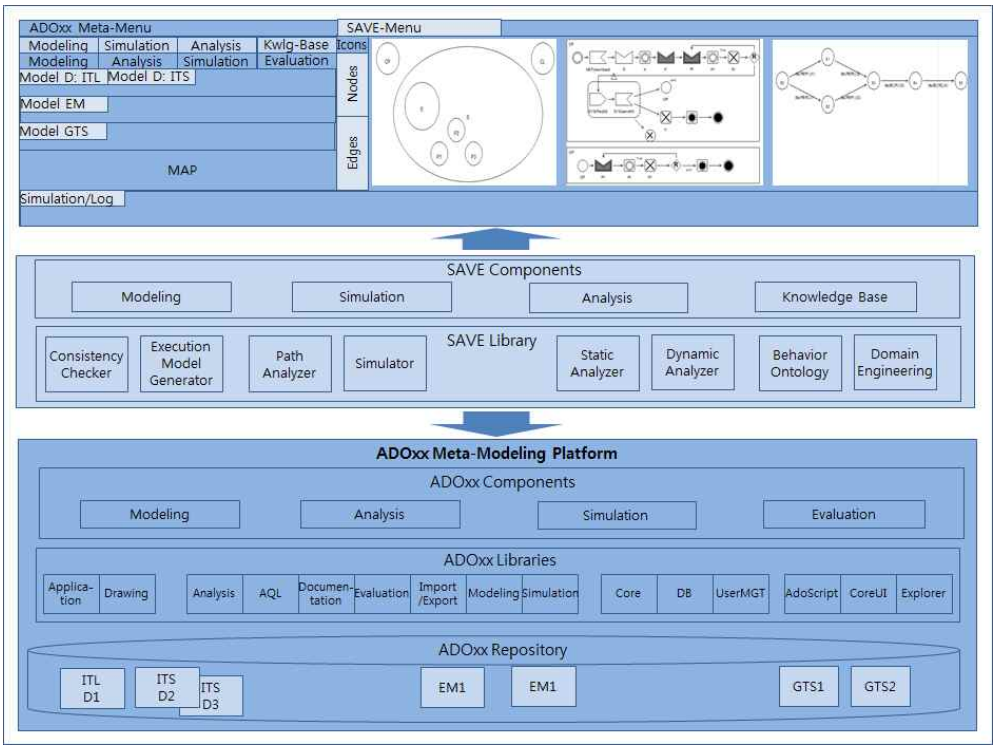

Figure 7. SAVE Tool on ADOxx

6 Tool

A tool, called SAVE (*Specification, Analysis, Verification and Evluation*), for δ-calculus has been developed on ADOxx meta-modeling platform (Fill, H. and Karagiannis, D., 2013), as shown in Figure 7. The tool consists of three basic components: Modeller, Simulator and Analyzer. Each components are based on its own graphical models, namely, ITL/ITS, Execution and GTS (*Geo-Temporal Space*)

models, respectively. The graphical representations of the models are designed by ADOxx Development Tool, and the procedures of the components are built from ADOxx libraries. Here in this paper only ITL and ITS models are presented. Figures 8 and 9 show the ITL and ITS views for the example in Section 4.

The first step of specification is to select the type of views. If an ITL view is selected as shown in Figure 8, its respective model is listed in Modelling component as shown in the figure. The ITL view is constructed with processes as nodes and their communications or movements as edges.

If a node, that is, process, is selected from the ITL view, its respective ITS view is selected in Modelling component as shown in Figure 9. The ITS view can be constructed with actions as nodes and transitions as edges.

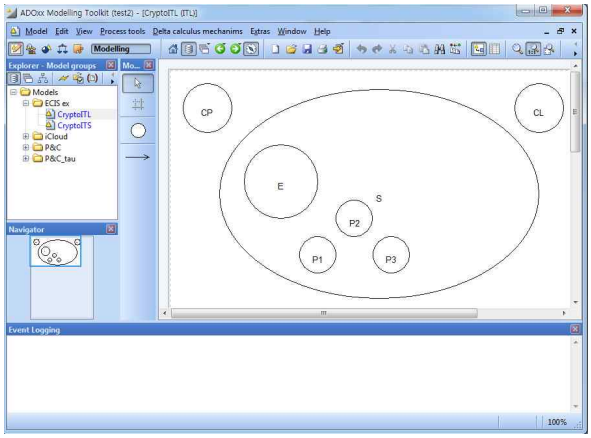

*Figure 8. ITL Example in **δ**-Editor*

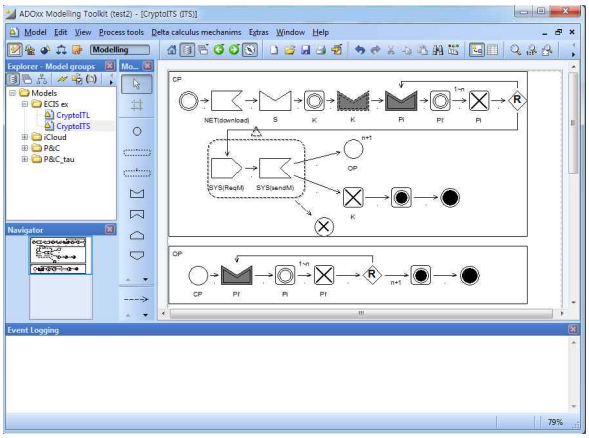

*Figure 9. ITS Example in **δ**-Editor*

7 Conclusions and Future Research

This paper presented δ-calculus and SAVE to represent the movements of business processes for distributed mobile enterprise applications with temporal constraints. Different from other candidate algebras, the calculus allows specification of various types of process movements over a conceptual geographical space. The properties of the movements, such as, autonomy/heteronomy, synchrony/asynchrony, priority and temporal boundary, make the target applications to be specified and analyzed in both geographical and temporal space for safety and security.

Future research includes developing analysis and verification methods and components for unsafe and insecure intrusions caused by other external processes.

Acknowledgment

This work was supported by Basic Science Research Programs through the NRF(*National Research Foundation of Korea*) funded by the Ministry of Education, Science and Technology(2010-0023787) and the MSIP(*Ministry of Science, ICT and Future Planning*), Korea, under the ITRC(*Information Technology Research Center*) support program (NIPA-2014-H0301-14-1023) supervised by the NIPA(*National IT Industry Promotion Agency*), 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).

References

R. Milner, Communication and Concurrency, Prentice hall, 1989

R. Singer and M. Teller, Process Algebra and the Subject-oriented Business Process Management Approach, *S-BPM ONE-Education and industrial Developments*, pp.135-150, 2012

Milner, R., J. Parrow and D. Walker. A calculus of mobile processes (i-ii).*Information and Computation*, 100 (1992), pp.1-77.

Cardelli, L., Gordon, A. Mobile Ambients. In: Nivat, M. (ed.) ETAPS 1998 and FOSSACS 1998. *LNCS*, vol. 1378, pp. 140–155. Springer, Heidelberg (1998).

R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, New York, 1982.

CryptoLocker Ransomware Inforamtion Guide and FAQ, www.bleepingcomputer.com.

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