이동 실시간 시스템의 명세를 위한 프로세스 대수

최영복, 최우림, 이문근
전북대학교 컴퓨터공학부
e-mail : {cyb0612, criscrowleopard}@gamil.com, moonkun@jbnu.ac.kr

Process Algebra for Specification of Mobile Real-time Systems

Yeong-Bok Choe, Woo-Rim Choi, Moon-Kun Lee
Dept. of Computer Engineering, Chonbuk National University

요 약

컴퓨터 시스템의 병렬, 분산, 이동, 실시간 적인 시스템들을 명세하기 위한 여러가지 정형기법들이 존재한다. 본 논문에서는 이동 실시간 시스템의 명세를 위한 정형기법으로서 δ-Calculus 를 정의하였다. δ-Calculus 의 가장 큰 특징은 프로세스의 이동성으로써 시간의 흐름에 따라 프로세스 간의 상호작용을 통해 프로세스가 이동하는 것을 표현할 수 있다. δ-Calculus 를 사용하여 프로세스의 이동성을 표현함으로써 시스템의 공간정보와 시간정보를 명세하고, 프로세스의 상태에 따른 보안적 특성을 나타낼 수 있다. 본 논문에서는 δ-Calculus 의 문법과 의미를 설명하고 이동성에 의한 특성을 분석하였다.

1. 서론

현대의 컴퓨터 시스템은 매우 방대하고 복잡하게 구성되어 있다. 이러한 시스템들은 수많은 프로세스들로 구성되며 각 프로세스들은 수시로 이동하고 메시지를 주고받는다.이와 같은 이동 실시간 시스템을 명세하고 검증하기 위해 π -Calculus[1], Mobile Ambient[2]와 같은 프로세스 대수[3]들이 제안되었다.

그러나 기존의 프로세스 대수들은 직접적인 이동성을 표현하는데 제약이 있다. 이러한 문제점을 해결하기 위하여 이동성을 표현하기 위한 프로세스 대수인 δ-Calculus 를 제안한다. δ-Calculus 는 프로세스의 이동성을 표현할 수 있는 프로세스 대수이며 이를 통해 공간적, 시간적 특성을 명세하고 분석할 수 있는 프로세스 대수이다.

2. δ-Calculus

2.1 Overview

δ-Calculus 는 이동 실시간 시스템의 명세를 위한 프로세스 대수이다. 이동성의 개념을 도입하여 시스템의 이동 이라는 속성을 명세할 수 있으며, 이를 통해 이동에 따른 시스템의 변화와 같은 공간적 특성을 표현할 수 있다.

δ-Calculus 의 이동은 모두 동기 방식으로 이루어진다. 따라서 특정 프로세스의 무분별한 이동으로 인한 시스템 문제를 미연에 방지할 수 있는 장점을 지니고 있다. 또한 특정한 프로세스에 대해서는 비동기적 이동을 허용할 수 있다.

2.2 Syntax

$P ::= 0$	Inaction
$\mid P \parallel P$	Parallel
$\mid P + P$	Choice
$\mid A(a,n)_t.P$	Action
$\mid P\backslash F$	Restriction
$\mid P[Q]$	Nesting
$\mid P \square^{a}_{[l,u]}(S,L,U,E)$	Scope
$\mid M{:}\,P$	Movement
$M ::= in_t\ P$	Move in the P
$\mid out_t\ P$	Move out of the P
$\mid P$ in	Permit the P to move in
$\mid P$ out	Permit the P to move out
$\mid get_t\ P$	Get the P in
$\mid put_t\ P$	Put the P out
$\mid P$ get	Permit the P to get in
$\mid P$ put	Permit the P to put out

(그림 1) **δ**-Calculus 의 Syntax

δ-Calculus 의 syntax 는 그림 1 과 같다.

(1) Inaction : 프로세스가 아무 행동도 하고 있지 않음을 나타낸다.

(2) Parallel : 두 프로세스가 동시에 병렬적으로 실행됨을 나타낸다.

(3) Choice : 두 프로세스 중 하나가 선택되어 실행된다.

(4) Action : n 의 우선도를 지닌 액션 a 를 t 시간 동안 통신채널 A 를 통해 실행한 후 프로세스 P 를 실행한다.

(5) Nesting : 프로세스 P 의 내부에 프로세스 Q 가 존재한다.

(6) Scope : 프로세스 P가 종료되는 시간에 따른 분기를 나타낸다. l은 time lower bound, u는 time upper bound 이다. 프로세스 P가 액션 a를 실행하며 종료하는 시간 t에 대하여 l < t < u인 경우 프로세스 S를, t < l 인 경우 프로세스 L을, u < t 인 경우 프로세스 U를 실행한다. 프로세스 P 실행 중 예외상황이 발생할 경우 프로세스 E를 실행한다.

(7) Movement : 프로세스의 이동을 나타낸다.

프로세스의 이동은 동기 방식이므로 이동의 요청과 그에 대한 허가 두 가지로 나뉠 수 있다. 또한 이동액션에 따라 실제 이동하는 프로세스가 달라진다. 이에 대한 구분은 표 1과같다.

<표 1> δ-Calculus 의 Movement

	특성	이동 프로세스
in P	Request(Active)	자신
out P	Request(Active)	자신
P in	Permit(Passive)	
P out	Permit(Passive)	
get P	Request(Active)	P
put P	Request(Active)	P
P get	Permit(Passive)	
P put	Permit(Passive)	

이들 중 Request 이동 액션에만 실행시간 t가 존재한다. Permit 이동 액션은 Request 가 올 때 까지 기다려야 하므로 정확한 실행시간을 정의할 수 없기 때문이다.

2.3 Semantics : Transition rules

δ-Calculus 에 대한 semantics 는 표 2와 같다.

<표 2> δ-Calculus 의 Semantics

In Action	$\dfrac{P \xrightarrow{Q\ in} P',\ Q \xrightarrow{in\ P} Q'}{P\|\|Q \xrightarrow{\delta} P'[Q']}(P\|\|Q)$
Out Action	$\dfrac{P \xrightarrow{Q\ out} P',\ Q \xrightarrow{out\ P} Q'}{P[Q] \xrightarrow{\delta} P'\|\|Q'}(P[Q])$
Get Action	$\dfrac{P \xrightarrow{get\ Q} P',\ Q \xrightarrow{P\ get} Q'}{P\|\|Q \xrightarrow{\delta} P'[Q']}(P\|\|Q)$
Put Action	$\dfrac{P \xrightarrow{put\ Q} P',\ Q \xrightarrow{P\ put} Q'}{P[Q] \xrightarrow{\delta} P'\|\|Q'}(P[Q])$
ScopeCT	$\dfrac{P \xrightarrow{A} P'}{P\square^{a}_{[l,u]}(S,L,U,E) \xrightarrow{A} P'\square^{a}_{[l-t,u-t]}(S,L,U,E)}$
ScopeE	$\dfrac{P \xrightarrow{a} P'}{P\square^{a}_{[l,u]}(S,L,U,E) \xrightarrow{a} S}(u > 0)$
ScopeTL	$\dfrac{P \xrightarrow{a} P'}{P\square^{a}_{[l,u]}(S,L,U,E) \xrightarrow{a} L}(l > 0)$
ScopeTU	$\dfrac{U \xrightarrow{b} U'}{P\square^{a}_{[l,u]}(S,L,U,E) \xrightarrow{b} U'}(u = 0)$
ScopeE	$\dfrac{E \xrightarrow{b} E'}{P\square^{a}_{[l,u]}(S,L,U,E) \xrightarrow{b} E'}(u > 0)$

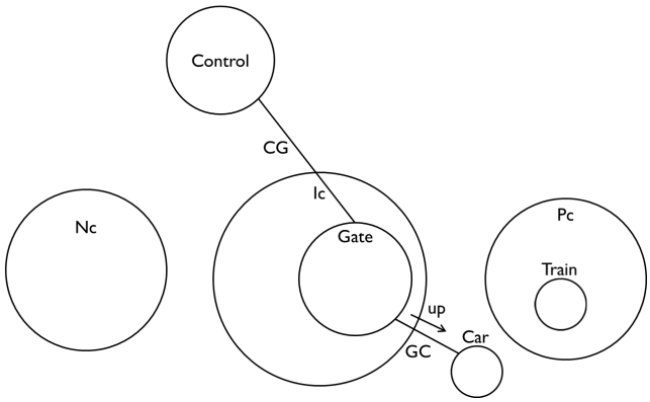
In Action, Out Action, Get Action, Put Action 은 이동 액션에 따른 프로세스의 동기 이동을 나타내고 있다. Out Action, Put Action 은 내부의 프로세스가 밖으로 나가는 것이므로 외부 프로세스가 나갈 프로세스를 포함하고 있어야 한다는 조건이 포함되며, In Action 과 Get Action 의 경우는 밖에서 안으로 들어와야 하므로 서로 평행관계에 있어야 한다는 조건이 붙는다.

ScopeCT 는 시간의 경과를 나타낸다. 액션 A가 수행되는데 t 시간이 소모되면 A가 수행된 후 각각의 time bound 에서 t 만큼 줄어든다. ScopeE 는 time bound 내에 P가 정상 종료된 경우이다. ScopeTL 은 lower time bound 전에 P가 종료되어 lower time out 프로세스로 넘어간 경우이다. ScopeTU 은 upper time bound 가 0 이므로 다음 상태에서는 자동적으로 upper time out 프로세스로 넘어갔음을 뜻한다. ScopeE 는 time bound 내에서는 언제든지 예외발생에 대한 처리가 가능함을 의미한다.

2.4 Graphical language

δ-Calculus 는 text 로 명세된 시스템을 시각적으로 이해하기 위한 그래픽 언어로 변환될 수 있다. 그래픽 언어는 In the Large(ITL)와 In the Small(ITS) 두 가지로 나뉘어 표현된다.

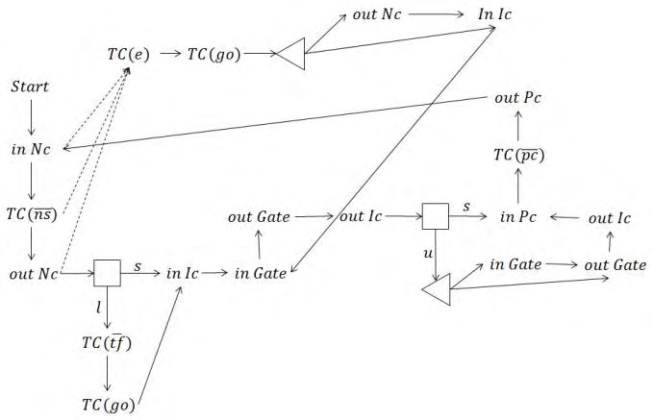
(1) ITL : 시스템 전체적인 관점에서 프로세스의 이동을 표현한다. 시간의 경과에 따라 프로세스가 이동하는 것을 직접적으로 표현한다. 그러나 ITL 에서는 프로세스의 이동 및 통신 채널만을 표현하므로 각각의 프로세스에 대한 자세한 실행과정은 표현하지 못한다. 그림 2는 ITL 명세의 예이다.



(그림 2) **δ**-calculus 의 ITL 명세

(2) ITS : 각각의 프로세스 관점에서 각 프로세스의 실행 단계를 표현한다. 각 액션의 수행 순서, Scope 에 의한 행동 분기 등을 자세하게

표현한다. 그러나 ITS 에서는 각각의 프로세스의 행동을 표현하므로 전체적인 프로세스의 움직임을 파악할 수 없다. 그림 3 은 ITS 명세의 예이다.



(그림 3) **δ**-calculus 의 ITL 명세

ITL 과 ITS 는 서로 표현하는 것이 다르며 따라서 어느 한가지의 명세만으로는 시스템의 행동을 명확히 표현하기 어렵다. 따라서 δ-Calculus 에서는 두 가지 모두를 사용하여 시스템을 명세하며, ITL 과 ITS 간의 연동을 통해 각각의 단점을 보완하며, 특정 시간에 대하여 프로세스의 전체적인 상태와 각 프로세스의 실행 단계를 표현할 수 있다.

3. 특성

3.1 이동

δ-Calculus 는 프로세스의 이동성을 표현하고 있으며 프로세스의 이동을 통해 공간적 특성을 표현한다. δ-Calculus 에서의 프로세스의 이동의 의미는 다음과 같다.

(1)실제적 이동 : 프로세스 또는 프로세스가 포함된 장치가 실제 공간적으로 이동하는 것을 의미한다. Railroad crossing system 등이 이에 해당한다.

(2)개념적 이동 : 프로세스의 실제 위치는 변하지 않지만 다른 프로세스간의 관계가 변함을 이동으로 표현한다. 프로세스의 부모-자식 간의 관계 변화, 프로세스의 상태 변화 등이 이에 해당한다.

실제 프로세스의 이동이 없더라도 개념적 이동의 도입을 통해 프로세스의 상태에 따른 특성 변화를 명세할 수 있다.

δ-Calculus 는 동기 이동을 통해 프로세스의 이동을 제어한다. 모든 프로세스는 허가 없이 이동할 수 없으며, 이동은 반드시 한 단계씩 이루어진다. 예를 들면 P[Q[R]]의 경우 R 이 P 밖으로 나오기 위해서는 반드시 Q 밖으로 나오는 과정을 거쳐야 하며, 따라서 P 와 Q 둘 모두의 허가가 있어야만 P 의 밖으로 나올 수 있다.

3.2 보안

δ-Calculus 는 이동의 특성을 통해 외부 환경에 영향을 받는 프로세스의 표현이 가능하다. 같은 프로세스라 하더라도 자신을 포함하는 프로세스가 있는가의 여부, 자신을 포함하는 프로세스가 어떤 프로세스인지 등의 상태에 따라 다르게 동작할 수 있다. 이는 특정 시스템의 명세를 분석할 시 단독으로는 안전한 시스템이더라도 외부 환경에 의하여 언제든지 불안전한 상태가 될 수 있음을 의미한다.

4. 비교분석

π-Calculus 는 Robin Milner 에 의해 개발되었으며 프로세스간의 통신을 주로 다루는 프로세스 대수이다. π-Calculus 는 통신 뿐만 아니라 value passing 을 통한 연결 채널의 변화를 통해 간접적인 이동을 표현하고 있다. 그러나 이동의 형태가 직접적으로 드러나지 않음으로 인하여 이동성의 측면에서 이해하는데 어려움이 있고 이동성의 표현이 매우 제한적이다.

Mobile ambient 는 프로세스의 이동을 표현하기 위한 프로세스 대수이다. Mobile ambient 는 비동기 이동을 도입하였으며 프로세스의 무분별한 이동에 제약을 가하기 위하여 Ambient 의 개념을 도입하였다. 그러나 모든 프로세스가 Ambient 를 통해 서로 연동하며 동작하므로 각각의 프로세스만을 보았을 시 어떻게 동작할 것인지를 파악하기 어렵다는 문제점이 있다.

δ-Calculus 는 직접적으로 이동을 표현할 뿐만 아니라 기존의 Text 기반 프로세스 대수와 유사한 형태로 이동성을 표현하고 있어 프로세스의 행동을 이해하기 수월하다. 또한 외부 환경에 따른 시스템의 안전성의 차이를 명확하게 표현할 수 있다는 장점이 있다. 실제 시스템의 안전성은 해당 시스템 자체만으로는 완벽하게 달성하기 어려우며 적절한 외부 환경이 갖추어져야 한다. 기존의 프로세스 대수로는 시스템 자체의 특성은 명세할 수 있으나 외부 환경과 프로세스간의 관계를 표현하는데 제약이 있었다. 그러나 δ-Calculus 를 통해 이를 명세하고 분석함으로써 안전성의 측면을 강화할 수 있다.

5. 예제

예제를 통하여 δ-Calculus 가 시스템을 어떻게 정의하고 분석하는지 살펴본다.

EMS(Emergency medical service)시스템은 환자를 병원으로 이송하는 시스템이다. EMS 시스템의 제약사항은 다음과 같다.

실행 제약사항

(1) 환자의 상태가 나빠져 병원에 연락이 온다.

(2) 병원은 환자에게 구급차를 보낸다.

(3) 구급차는 환자의 위치로 이동하여 환자를 이송한다.

1) 병원과 위치까지의 거리는 10 분이다.

2) 위치에 도착 후 환자를 3 분 내에 구급차로 옮긴다.

(4) 환자가 병원에 도착하면 병원은 환자를 치료한다.

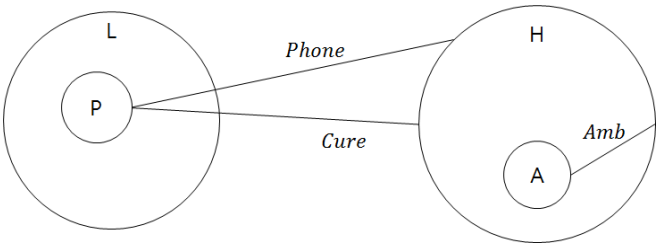
그림 4는 이에 대한 δ-Calculus 명세를 나타낸다.

$$
\begin{aligned}
&\text{EMS} ::= \text{L[P]} \;||\; \text{H[A]}\backslash\{\text{call}, \text{go}, \text{cure}\}\\
&\text{H} ::=\\
&\text{Phone}(\text{call}, 1)_1.\, \text{Amb}(\overline{\text{go}}, 1)_1.\, \text{Cure}(\overline{\text{cure}}, 1)_{60}.\, \text{H} \;||\; \text{HM}\\
&\text{HM} ::= \text{A in: HM} \;||\; \text{A out: HM}\\
&\text{L} ::= \text{A in: L} \;||\; \text{A out: L}\\
&\text{P} ::= \text{Phone}\left(\overline{\text{call}}, 1\right)_1.\, \text{A get: A put: Cure}(\text{cure}, 1)_{60}\\
&\text{A} ::= \text{Amb}(\text{go}, 1)_1.\, \text{out}_1\ \text{H: in}_{10}\ \text{L: get}_3\ \text{P: out}_{10}\ \text{L}\\
&\qquad : \text{in}_1\ \text{H: put}_3\ \text{P}
\end{aligned}
$$

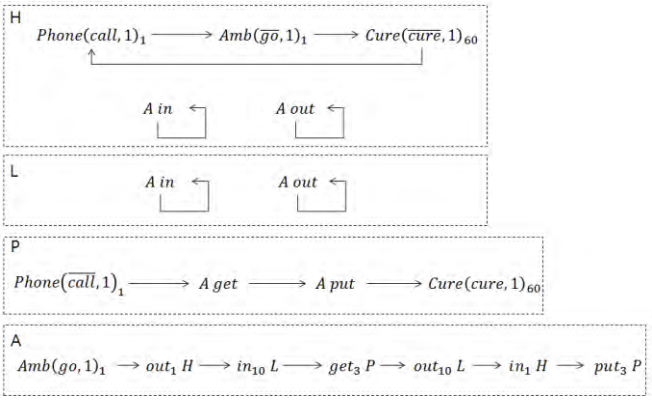
(그림 4) EMS 시스템의 **δ**-Calculus 명세

명세 중 HM, L은 프로세스 A에 대한 이동 허가를 무한히 반복한다. 이를 통해 프로세스 A에 한해서는 H, L에 대한 비동기적 이동이 가능하도록 한다.

위의 text 명세를 기반으로 ITL, ITS 명세를 함으로써 시각적인 표현을 나타낸다. 그림 5, 6은 각각 ITL, ITS 명세를 나타낸다.



(그림 5) EMS 시스템의 ITL 명세

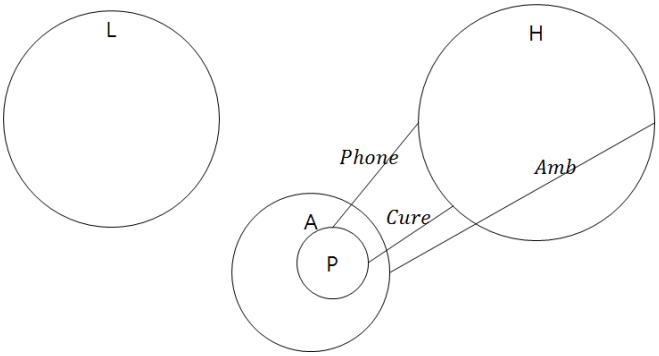


(그림 6) EMS 시스템의 ITS 명세

시간의 흐름에 따라 EMS 시스템의 행동을 살펴보면 다음과 같다.

(1) t = 1 : 병원이 환자의 연락을 받음
(2) t = 2 : 구급차가 병원의 지시를 받음
(3) t = 3 : 구급차가 병원 밖으로 나옴
(4) t = 13 : 구급차가 환자의 위치로 이동
(5) t = 16 : 환자를 구급차로 옮김
(6) t = 26 : 구급차가 병원으로 이동
(7) t = 27 : 구급차가 병원 안으로 이동
(8) t = 30 : 환자를 병원으로 옮김
(9) t = 90 : 환자의 치료가 끝남

위와 같은 시간 흐름에 따라 ITL 명세에서 프로세스들이 이동을 하게 된다. 그림 7은 17 < t < 26 시점에서의 ITL 명세를 나타낸다.



(그림 7) EMS 시스템의 ITL 명세(17 < t < 26)

6. 결론

본 논문은 이동성을 표현하기 위한 프로세스 대수인 δ-Calculus를 제안하였다. 기존의 프로세스 대수들과 달리 직접적인 공간적 이동 개념을 도입함으로써 이동 실시간 시스템을 위한 시간 및 공간 속성에 대한 정의를 통해 다양한 시스템의 명세를 수행하고, 시간에 따른 프로세스의 이동과 프로세스의 위치에 따른 보안적 특성에 대해 명세할 수 있었다.

ACKNOWLEDGEMENT

이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원(No.2010-0023787) 과 미래창조과학부 및 정보통신산업진흥원의 대학IT연구센터육성 지원사업/IT융합고급인력과정지원사업(NIPA-2014-H0301-14-1023)의 연구 결과로 수행되었음.

참고문헌

[1] Milner, R., J. Parrow and D. Walker, “A calculus of mobile processes (i-ii)”, Information and Computation 100 (1992), pp.1-77
[2] Cardelli, L., Gordon, A. “Mobile Ambients”, In: Nivat, M. (ed.) ETAPS 1998 and FOSSACS 1998. LNCS, vol. 1378, pp. 140–155. Springer, Heidelberg (1998)
[3] Jos C. M. Baeten. “A Brief History of Process Algebra.” Report CSR 04-02, Eindhoven University of Technology, 2004.