情報処理学会論文誌　プログラミング　Vol.10　No.4　12–27 (July 2017)

物流記述のための混合型時間アンビアント計算

藤坂 吉秀[1]　稲森 啓太[1]　樋口 昌宏[1,a)]　加藤 暢[1,b)]

受付日 2016年12月9日, 採録日 2017年2月17日

概要：本論文では，Ambient Calculus（AC）を時間的な制約が記述できるよう拡張した混合型時間アンビアント計算（HTAC）と，それを用いた物流計画記述法を提案する．HTAC は AC のケーパビリティに加えて，有効期限付きケーパビリティと待機ケーパビリティを持つ．これにより，AC の特徴であった階層構造の動的な変化の表現に加え，タイムアウト動作や 2 つのイベント間の発生間隔の上下限の指定などの時間制約が指定可能となる．有効期限付きケーパビリティを連続するイベントが離散的に発生するオブジェクトの移動の表現に用い，AC のケーパビリティを瞬時に発生するべきオブジェクトの移動情報の通知などに用いることで，各種交通機関を用いた人の移動や海上物流などの物流計画が記述できる．このようなケーパビリティを使い分ける基準として，プロセス式が物流計画の記述と見なせるための条件を与える．また，HTAC プロセス式から実行可能なイベントを選択し，式を遷移させる処理系を開発した．この処理系を用いて，プロセス式が意図どおりの動作をするかなどの確認を行うことができる．その処理系を活用した物流監視システムの構築方法について述べる．物流監視システムは HTAC 式による物流計画に違反したオブジェクトの移動を検出するとともに，HTAC 式に沿った移動を誘導するようなシステムである．

キーワード：形式仕様記述，プロセス代数，移動型プロセス，時間制約

Hybrid Timed Ambient Calculus for Logistics Specification

Yoshihide Fujisaka[1]　Keita Inamori[1]　Masahiro Higuchi[1,a)]　Toru Kato[1,b)]

Received: December 9, 2016, Accepted: February 17, 2017

Abstract: In this paper, we propose the hybrid timed ambient calculus (HTAC) as an extension of the ambient calculus (AC). We also propose a way for specifying logistics plans using HTAC. In addition to the capabilities of AC, HTAC contains expiration time and waiting capabilities that enable us to specify timeout–like behaviors and specify the lower or upper bounds of the timing interval between two events. Therefore, we can represent logistics specifications such as human travel plans with various transportation systems or maritime logistics. In HTAC logistics specifications, we use the expiration time capabilities for physical movements, which occur discretely, and the original AC capabilities for the physical movement notifications, which occur simultaneously. We give a condition for the HTAC formula, which is regarded as a logistics specification. We developed a formula processing system, which can reduce a process formula by selecting executable events, enabling us to confirm whether the process formula behaves as intended. We will describe the method to build a logistics monitoring system with the formula processing system. The system can detect the physical movement that violates the logistics plan, or can guide the movements in compliance with the logistics plan.

Keywords: formal specification, process algebra, mobile process, timing constraint

1. はじめに

アンビアント計算 [4]（以下，AC と呼ぶ）は，Microsoft Research の Luca Cardelli と Andrew D. Gordon によって開発されたプロセス代数であり，アンビアント構文の再帰的な利用によりオブジェクトの階層構造を記述し，階層構造を変化させるための構文と，それに関する遷移規則により，動的な階層構造を持つシステムを形式的に記述することができる言語である．我々は AC による物流計画の記

[1] 近畿大学
Kindai University, Higashi-Osaka, Osaka 577–8502, Japan
[a)] higuchi@info.kindai.ac.jp
[b)] kato@info.kindai.ac.jp

述とそれに基づく物流監視システムの構築を提案している [7], [10], [11].

AC を含め，プロセス代数を用いて対象となるシステムをモデル化する場合，式の遷移は記述対象となっているシステムにおいて瞬時的に実行されるイベントに対応するものと解釈され，実行可能なイベント系列の集合によってシステムの振舞いを指定するものと考える．通常のプロセス代数では実時間を記述する枠組みを備えていないため，イベント間の時間間隔は考慮されない．しかし，物流のような複数のオブジェクトの移動が離散的に発生するシステムを表現しようとすると，オブジェクトの移動や積み込みに一定の時間を要するなどの時間制約を適切に表現する必要がある．

そこで本論文では，AC を時間拡張した混合型時間アンビアント計算（以下，HTAC と呼ぶ）と，HTAC を用いた物流計画記述を提案する．HTAC では，AC のケーパビリティに加え，期限切れで無効化する有効期限付きのケーパビリティと，時間付き π 計算 [14] 同様に指定した時間だけ待機するケーパビリティを導入する．Timed Mobile Ambient [1]（以下，tMA と呼ぶ）でもケーパビリティが期限切れによって無効化するが，7 章で述べるように HTAC のケーパビリティとは異なる動作である．

ケーパビリティの追加により，AC の特徴であった階層構造の動的な変化の表現に加え，タイムアウト動作を特別な構文を用意することなく指定することや，2 つのイベント間の発生間隔の上下限を指定することが可能となる．タイムアウト動作の指定により，一定時間指定したイベントを行わなければ別のイベントへの変更を指定することができ，イベント間の発生間隔の指定により，オブジェクトの移動時間などを指定できるようになる．

2 章では時間拡張を行った HTAC の構文規則や遷移規則について述べる．また，特別なアンビアントとして，指定した時間内に指定したケーパビリティが消費されなかったことを示すアンビアントと，内部遷移を禁止することで不必要となったプロセスを破棄するためのアンビアントを導入する．3 章では，HTAC を用いたタイムアウト動作やイベント間の時間間隔の指定などの時間に依存した振舞いの記述方法について述べ，それらを記述する際に用いるマクロ記法を示す．4 章では，HTAC を用いた物流計画記述として，有効期限付きケーパビリティはオブジェクトの階層構造を変動させるために用い，AC のケーパビリティは物理的な階層の変動が発生したことを通知するためなどに用いることについて述べる．また，これらの使い分けの基準を，プロセス式が物流計画記述と見なせるための条件として定義する．5 章では，物流計画記述と見なせるための条件を満たした HTAC 式を遷移させる処理系について述べ，この処理系を用いた検査実験について述べる．6 章では，HTAC を用いた物流計画記述に基づく物流監視システムの構築方法について述べる．7 章では HTAC と関連研究についての比較を行う．

2. 混合型時間アンビアント計算

2.1 アンビアント計算

AC とは，動的な階層構造を持つ複数のコンポーネントからなるシステム（以降「系」と呼ぶ）を記述するための言語である．AC では，各コンポーネントをアンビアントという互いに出入りが可能な「境界を持った場」で表現する．また，階層構造を変化させるイベントとして enter ケーパビリティ，exit ケーパビリティ，境界の消滅を表す open ケーパビリティの 3 種類を持つ．これらのケーパビリティを消費することにより，プロセス式は異なる階層構造を持つプロセス式に遷移する．AC を用いることで，現在の系の階層構造とそこから可能なイベント系列の集合を指定することができる．これにより，モバイルエージェント群の振舞いの指定や物流計画の記述が可能となる．

AC では，モバイルエージェントそのものやその実行環境，物流における貨物やトラックといったオブジェクトなど，系内の各コンポーネントをアンビアントで表現する．また，そのコンポーネントの移動情報を系内の別のコンポーネントに通知するなどにもアンビアントを用いる [5], [8]．本論文では，移動情報の通知などに用いられるアンビアントを制御アンビアントと呼び，これらの移動など，系の動作そのものを表現していない遷移を制御遷移と呼ぶ．このため AC の遷移系列から制御遷移を消去したものを系のイベント系列と考える．

2.2 構文規則と遷移規則

HTAC では，文献 [4] における階層構造の変化を指定するケーパビリティに有効期限付きのもの，および指定した時間だけ待機することを表すものを追加している．

2.2.1 構文規則

有効期限および待機時間の長さを表現するため正の整数値を用い，有効期限が無限であることを表すために ∞ を用いる．HTAC は文献 [4] に従った AC の構文規則と合わせた以下の構文規則を持つ．

定義 2.1（構文規則）

$P, Q ::=$	processes
$(\nu\ n)P$	restriction
0	inactivity
$P \mid Q$	composition
$!P$	replication
$M[P]$	ambient
$M.P$	capability action
$(x).P$	input action
$\langle M \rangle$	async output action

$M, N ::=$	capabilities
x	variable
n	name
$in\ M$	enter into M immediately
$out\ M$	exit out of M immediately
$open\ M$	open M immediately
$in(t_\infty)\ M$	can enter into M within t_∞ time unit
$out(t_\infty)\ M$	can exit out of M within t_∞ time unit
$open(t_\infty)\ M$	can open M within t_∞ time unit
$wait(t)$	wait t time unit
ε	null
$M.N$	path
$t ::=$	time
$1, 2, \ldots$	positive integer
$t_\infty ::=$	time with infinity
t	time
∞	infinity

□

通常プロセス式中の「(」,「)」,「0」は混乱のない範囲で省略される．$in\ M$，$out\ M$，$open\ M$ を通常のケーパビリティと呼び，$in(t_\infty)\ M$，$out(t_\infty)\ M$，$open(t_\infty)\ M$ を有効期限付きケーパビリティと呼び，$wait(t)$ を待機ケーパビリティと呼ぶ．

ambient 構文を用いることで内部にプロセスを持つオブジェクト（以降アンビアントと呼ぶ）を表現することができ，これを再帰的に用いることによりアンビアント間の入れ子構造を表現できる．たとえば $l[m[0] \mid n[p[0]]]$ によりアンビアント l 中にアンビアント m，n が存在し，さらに n 中に p が存在するという階層構造を表現できる．

定義 2.2（活性アンビアントと活性ケーパビリティ）
プロセス式 P 中で，capability action 構文を用いて何らかのケーパビリティに先行されている部分プロセス式 P' 中のアンビアントおよびケーパビリティを非活性と呼び，そうでないものを活性と呼ぶ．□

たとえばプロセス式 $n[in\ m.out\ m.p[\,]] \mid m[\,]$ で，$out\ m$ は非活性ケーパビリティ，$p[\,]$ は非活性アンビアントであり，それら以外は活性である．プロセス式 P において，非活性ケーパビリティ，非活性アンビアントは P からの直接の遷移に関与することはない．

AC 同様である HTAC の構造合同を以下のように定義する．

定義 2.3（構造合同）

$P \equiv P$	(Refl)
$P \equiv Q \Rightarrow Q \equiv P$	(Symm)
$P \equiv Q,\ Q \equiv R \Rightarrow P \equiv R$	(Trans)
$P \equiv Q \Rightarrow (\nu\ n)P \equiv (\nu\ n)Q$	(Res)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Par)
$P \equiv Q \Rightarrow\ !P \equiv\ !Q$	(Repl)
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$	(Amb)
$P \equiv Q \Rightarrow M.P \equiv M.Q$	(Action)
$P \mid Q \equiv Q \mid P$	(Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Par Assoc)
$!P \equiv P \mid !P$	(Repl Par)
$(\nu\ n)(\nu\ m)P \equiv (\nu\ m)(\nu\ n)P$	(Res Res)
$(\nu\ n)(P \mid Q) \equiv P \mid (\nu\ n)Q$ if n が P の自由 name でない	(Res Par)
$(\nu\ n)(m[P]) \equiv m[(\nu\ n)P]$ if $n \neq m$	(Res Amb)
$P \mid 0 \equiv P$	(Zero Par)
$(\nu\ n)0 \equiv 0$	(Zero Res)
$!0 \equiv 0$	(Zero Repl)
$P \equiv Q \Rightarrow (x).P \equiv (x).Q$	(Input)
$\varepsilon.P \equiv P$	(ε)
$(M.M').P \equiv M.M'.P$	(.)

□

定義 2.3 および以降では「⇒」は含意を表す．たとえば $l[m[0] \mid n[p[0]]]$ と $l[n[p[0]] \mid m[0]]$ は同一視する．

AC 同様，HTAC では，「$Name \triangleq P$」というようなプロセス定義式によりプロセス式に名前を与えておき，別のプロセス式中に $Name$ と書くことでプロセス式を呼び出すことができる．一般には自分自身を呼び出すような再帰的なプロセス定義式も許される．しかし $Name \triangleq P \mid Name'$ や $Name \triangleq P \mid M[Name']$（$Name'$ はプロセス呼び出し）のように，何らのケーパビリティにも先行されないプロセス呼び出しがプロセス定義式右辺に出現することを禁止する．

2.2.2　遷移規則

HTAC ではアンビアント内に記述されたケーパビリティにより，アンビアント階層の動的な変化を記述する．すなわち，ケーパビリティを消費することによりプロセス式は異なる階層構造を持つプロセス式に遷移する．プロセス式の遷移について，通常のケーパビリティの消費による遷移関係「$\rightarrow$」，有効期限付きケーパビリティによる遷移関係「$\xrightarrow{\delta}$」，そして時間経過による遷移関係「$\xrightarrow{\tau}$」を以下のように定義する．

定義 2.4（通常のケーパビリティによる遷移規則）

$N[in\ M.P \mid Q] \mid M[R] \rightarrow M[N[P \mid Q] \mid R]$	(In)
$M[N[out\ M.P \mid Q] \mid R] \rightarrow N[P \mid Q] \mid M[R]$	(Out)
$open\ M.P \mid M[Q] \rightarrow P \mid Q$	(Open)
$P \rightarrow Q \Rightarrow (\nu\ n)P \rightarrow (\nu\ n)Q$	(Res)
$P \rightarrow Q \Rightarrow N[P] \rightarrow N[Q]$	(Amb)
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$	(Par)
$P' \equiv P,\ P \rightarrow Q,\ Q \equiv Q' \Rightarrow P' \rightarrow Q'$	(Cong)

□

(In) のような遷移を「N enter M」，(Out) のような遷移を

「N exit M」，(Open) のような遷移を「M disappear」と呼ぶ．(In)，(Out)，(Open) の左辺のケーパビリティ「$in\ M$」，「$out\ M$」，「$open\ M$」を消費可能と呼ぶ．また，「$\rightarrow$」の反射推移的閉包を「$\xrightarrow{*}$」と書く．

定義 2.5（**安定なプロセス式**）　$P \rightarrow Q$ なる Q を持たないプロセス式 P は安定であるという．　□

以降，安定なプロセス式を表す記号として $\mathbf{P}$ のような字体を用いることにする．

定義 2.6（**有効期限付きケーパビリティによる遷移規則**）安定なプロセス式からの遷移「$\xrightarrow{\delta}$」を以下のように定義する．

$$N[in(t_\infty)\ M.P \mid Q] \mid M[R] \xrightarrow{\delta} M[N[P \mid Q] \mid R] \quad \text{(In-p)}$$
$$M[N[out(t_\infty)\ M.P \mid Q] \mid R] \xrightarrow{\delta} N[P \mid Q] \mid M[R] \quad \text{(Out-p)}$$
$$open(t_\infty)\ M.P \mid M[Q] \xrightarrow{\delta} P \mid Q \quad \text{(Open-p)}$$
$$\mathbf{P} \xrightarrow{\delta} Q \Rightarrow (\nu\ n)\mathbf{P} \xrightarrow{\delta} (\nu\ n)Q \quad \text{(Res-p)}$$
$$\mathbf{P} \xrightarrow{\delta} Q \Rightarrow N[\mathbf{P}] \xrightarrow{\delta} N[Q] \quad \text{(Amb-p)}$$
$$\mathbf{P} \xrightarrow{\delta} Q \Rightarrow \mathbf{P} \mid \mathbf{R} \xrightarrow{\delta} Q \mid \mathbf{R} \quad \text{(Par-p)}$$
$$\mathbf{P}' \equiv \mathbf{P},\ \mathbf{P} \xrightarrow{\delta} Q,\ Q \equiv Q' \Rightarrow \mathbf{P}' \xrightarrow{\delta} Q' \quad \text{(Cong-p)}$$

□

(In-p) のような遷移を「N Enter M」，(Out-p) のような遷移を「N Exit M」，(Open-p) のような遷移を「M Disappear」と呼ぶ．また，(In-p)，(Out-p)，(Open-p) の左辺のケーパビリティ「$in(t_\infty)\ M$」，「$out(t_\infty)\ M$」，「$open(t_\infty)\ M$」を消費可能と呼ぶ．

定義 2.7（**時間経過による遷移規則**）　安定なプロセス式からの遷移「$\xrightarrow{\tau}$」を以下のように定義する．

$$in\ M.P \xrightarrow{\tau} in\ M.P \quad \text{(In-t)}$$
$$out\ M.P \xrightarrow{\tau} out\ M.P \quad \text{(Out-t)}$$
$$open\ N.P \xrightarrow{\tau} open\ N.P \quad \text{(Open-t)}$$
$$in(\infty)\ M.P \xrightarrow{\tau} in(\infty)\ M.P \quad \text{(In-tp1)}$$
$$in(t)\ M.P \xrightarrow{\tau} in(t-1)\ M.P\ (t \geq 2) \quad \text{(In-tp2)}$$
$$in(1)\ M.P \xrightarrow{\tau} 0 \quad \text{(In-tp3)}$$
$$out(\infty)\ M.P \xrightarrow{\tau} out(\infty)\ M.P \quad \text{(Out-tp1)}$$
$$out(t)\ M.P \xrightarrow{\tau} out(t-1)\ M.P\ (t \geq 2) \quad \text{(Out-tp2)}$$
$$out(1)\ M.P \xrightarrow{\tau} 0 \quad \text{(Out-tp3)}$$
$$open(\infty)\ M.P \xrightarrow{\tau} open(\infty)\ M.P \quad \text{(Open-tp1)}$$
$$open(t)\ M.P \xrightarrow{\tau} open(t-1)\ M.P\ (t \geq 2) \quad \text{(Open-tp2)}$$
$$open(1)\ M.P \xrightarrow{\tau} 0 \quad \text{(Open-tp3)}$$
$$wait(t).P \xrightarrow{\tau} wait(t-1).P\ (t \geq 2) \quad \text{(Wait-1)}$$
$$wait(1).P \xrightarrow{\tau} P \quad \text{(Wait-2)}$$
$$\mathbf{P} \xrightarrow{\tau} Q \Rightarrow (\nu\ n)\mathbf{P} \xrightarrow{\tau} (\nu\ n)Q \quad \text{(Res-t)}$$
$$\mathbf{P} \xrightarrow{\tau} Q \Rightarrow !\mathbf{P} \xrightarrow{\tau} !Q \quad \text{(Repl-t)}$$
$$\mathbf{P} \xrightarrow{\tau} Q \Rightarrow N[\mathbf{P}] \xrightarrow{\tau} N[Q] \quad \text{(Amb-t)}$$
$$\mathbf{P} \xrightarrow{\tau} Q, \mathbf{P}' \xrightarrow{\tau} Q' \Rightarrow \mathbf{P} \mid \mathbf{P}' \xrightarrow{\tau} Q \mid Q' \quad \text{(Par-t)}$$
$$\mathbf{P}' \equiv \mathbf{P},\ \mathbf{P} \xrightarrow{\tau} Q,\ Q \equiv Q' \Rightarrow \mathbf{P}' \xrightarrow{\tau} Q' \quad \text{(Cong-t)}$$

□

(In-tp2)，(Out-tp2)，(Open-tp2) は時間経過により，活性である有効期限付きケーパビリティの有効期限が 1 単位時間短くなったことを表す．また，(In-tp3)，(Out-tp3)，(Open-tp3) は有効期限付きケーパビリティの期限切れ，すなわち期限の切れたケーパビリティとそれに引き続く部分式 P が無効化されることを表している．(Wait-1) は活性である待機ケーパビリティの待機時間が 1 単位時間短くなったことを表し，(Wait-2) は待機時間満了により待機ケーパビリティが消費されて，引き続く P が活性化されることを表している．また (Par-t) は並行な位置にあるプロセス間で時刻が同期して進むことを表している．これらは時間付き π 計算 [14] の時間経過アクションと同じものである．これらによりイベント間に時間経過が挿入可能となり，時間経過をともなう系の記述が可能となる．

$P = \mathbf{P_0} \xrightarrow{\tau} \mathbf{P_1}, \mathbf{P_1} \xrightarrow{\tau} \mathbf{P_2}, \ldots, \mathbf{P_{n-1}} \xrightarrow{\tau} P_n = Q$ であるとき $P \xrightarrow[(n)]{\tau} Q$ と書く．これは，他の有効期限付きケーパビリティや通常のケーパビリティが消費されることなく n 単位時間経過し，P から Q へ遷移することを表す．

「$\xrightarrow{\delta}$」および「$\xrightarrow{\tau}$」が安定な式のみに適用できるという条件は，通常のケーパビリティによる遷移が有効期限付きケーパビリティによる遷移に優先し，消費可能なものは瞬時的に実行しなければならないことを表している．一方，有効期限付きケーパビリティは消費可能であっても，必ずしも消費する必要はなく，時間経過することも可能である．

2.3　特別なアンビアントとそれにともなう遷移規則の変更

本論文では HTAC による系の記述に際して，特別なアンビアント名として *alarm* と *trashbox* を導入する．通常のアンビアントの名前としてこれらを用いることはできない．

2.3.1　*alarm* アンビアント

HTAC による系の仕様記述では，有効期限付きケーパビリティは消費可能であっても消費せずに時間経過が可能である．このため，イベントが実行可能であるにもかかわらず永遠に実行しないようなイベント系列を許容する系しか記述できない．そのようなイベント系列を排除するため *alarm* という名前を持つ特別なアンビアントを導入する．

定義 2.8（**終末的プロセス**）　プロセス式 P において *alarm* アンビアントが活性であるとき，P は終末的であるという．　□

消費可能な有効期限付きケーパビリティが放置された際に，*alarm* アンビアントが活性化するように記述する．これにより，あるプロセス式の表すイベント系列の集合から，終末的プロセスへ至るイベント系列を除外する．また，終末的プロセスからの遷移はない．*alarm* アンビアントの用い方は 3.2 節で述べる．

2.3.2　*trashbox* アンビアント

HTAC による系の仕様記述では，様々な場合分けなどを

考慮するため，必ずすべての部分式が用いられるわけではない．この場合，無用となった部分式が，他の部分式に副作用として意図しない影響を与えてしまうことが起こりうる．これを防ぐため，用いられなかった部分式を破棄する特別なアンビアントとして *trashbox* アンビアントを導入する．移動しない *trashbox* アンビアント内に不要となった部分式を格納し，*trashbox* 内部での遷移を禁止することで副作用を防ぐ．*trashbox* 内部での遷移を禁止するため *trashbox* という名前のアンビアントに対して一部遷移規則に制限を加える．

定義 2.9（制限後の遷移規則）

$$N[in\ M.P \mid Q] \mid M[R] \to M[N[P \mid Q] \mid R \quad \text{if } N \text{ が } trashbox \text{ でない} \quad \text{(In)}$$

$$M[N[out\ M.P \mid Q] \mid R] \to N[P \mid Q] \mid M[R] \quad \text{if } M \text{ または } N \text{ が } trashbox \text{ でない} \quad \text{(Out)}$$

$$open\ M.P \mid M[Q] \to P \mid Q \quad \text{if } M \text{ が } trashbox \text{ でない} \quad \text{(Open)}$$

$$P \to Q \Rightarrow N[P] \to N[Q] \quad \text{if } N \text{ が } trashbox \text{ でない} \quad \text{(Amb)}$$

$$N[in(t_\infty)\ M.P \mid Q] \mid M[R] \xrightarrow{\delta} M[N[P \mid Q] \mid R \quad \text{if } N \text{ が } trashbox \text{ でない} \quad \text{(In-p)}$$

$$M[N[out(t_\infty)\ M.P \mid Q] \mid R] \xrightarrow{\delta} N[P \mid Q] \mid M[R] \quad \text{if } M \text{ または } N \text{ が } trashbox \text{ でない} \quad \text{(Out-p)}$$

$$open(t_\infty)\ M.P \mid M[Q] \xrightarrow{\delta} P \mid Q \quad \text{if } M \text{ が } trashbox \text{ でない} \quad \text{(Open-p)}$$

$$P \xrightarrow{\delta} Q \Rightarrow N[P] \xrightarrow{\delta} N[Q] \quad \text{if } N \text{ が } trashbox \text{ でない} \quad \text{(Amb-p)}$$

$$P \xrightarrow{\tau} Q \Rightarrow N[P] \xrightarrow{\tau} N[Q] \quad \text{if } N \text{ が } trashbox \text{ でない} \quad \text{(Amb-t)}$$

□

この制限により，(i) *trashbox* という名前のアンビアントは移動しない．(ii) *trashbox* 内でプロセス間の相互作用は起こらない．(iii) *trashbox* からの exit，*trashbox* の disappear は発生しない．これは *trashbox* 内のプロセスは遷移には影響を与えないことを示している．

この制約は Typed Ambient Calculus [3]（以下，Typed AC と呼ぶ）の “型” の概念に類似している．Typed AC ではアンビアントの名前をグループ化し，グループ間の作用を制限することで外的要因からの意図しない影響を防いでいる．制約を受けた *trashbox* について時間経過以外を Typed AC に沿った型情報で記述すると以下のようになる．

$$(\nu\ trashbox : Trsh^{\frown}\emptyset[^{\frown}\emptyset, ^{\circ}\emptyset, Shh])$$

直観的にこの式は，*trashbox* という名前が *Trsh* グループに属し，この名前を持つアンビアントは，任意の名前 m に対し，objective move（$go(in\ m)$ など）による移動ができず（1 つ目の $^{\frown}\emptyset$），$in\ m$ など通常のケーパビリティによる移動ができず（2 つ目の $^{\frown}\emptyset$），内部で $open\ m$ を許さず（$^{\circ}\emptyset$），内部でいっさいの入出力を許さない（*Shh*）ことを表す．

また，外的要因からの意図しない影響を防ぐ構文を含んだものとして Safe Ambients [8] も存在する．Typed AC では，グループそれぞれに型情報を与え，どのアンビアントがどのグループに属するかを明記する必要がある．また Safe Ambients では出入りされる側のアンビアントにもその動作を許可するかどうかを明記する必要がある．これに対し HTAC では，*trashbox* アンビアントに関する制約を遷移規則内に設けることにより，制限する範囲などをプロセス式の中で考慮しなくても，内部に格納することで外的要因による意図しない影響を防ぐことができる．具体例については 4.5 節で述べる．

2.4　HTAC による記述例

HTAC により動的な階層構造を持つ系を記述した簡単な例を示す．下記の式はクライアントサーバシステムにおけるデータリクエストを表現したものである．これはクライアント（Clnt）がリクエストメッセージ（ReqM）を送り，サーバ（Srvr）が受け取ると，サーバがデータ（Dt）をサーバからクライアントに送るという動作を記述している．

上記の 4 つのアンビアントのような直接の記述対象を表すアンビアントではないアンビアントが，2.1 節で述べた制御アンビアントである．

$$\begin{array}{l} Clnt[\\ \quad REQM \mid open\ arrvD.open\ x \\ ] \mid Srvr[\\ \quad !open\ arrvM.Dt[out\ Srvr.in(5)\ Clnt.arrvD[out\ Dt]] \\ ] \end{array}$$

リクエストメッセージの送信はプロセス定義式で *REQM* で以下のように定義している．

$$\begin{array}{l} REQM \triangleq \\ \quad (ReqM[out\ Clnt.in(5)\ Srvr.arrvM[out\ ReqM]] \\ \quad \mid wait(10).open\ x.REQM \\ \quad \mid x[\,]) \end{array}$$

arrvM アンビアントはメッセージの到着をサーバに伝え，*arrvD* アンビアントはデータの到着をクライアントに伝えるための制御アンビアントである．x アンビアントはリクエストメッセージ送信後 10 単位時間経過，もしくは，10 単位時間以内にデータを受け取ると *open* される制御アンビアントである．この記述のイベント系列の 1 つは「*ReqM* exit *Clnt*」,「*ReqM* Enter *Srvr*」,「*Dt* exit *Srvr*」,「*Dt* Enter *Clnt*」となる．もしいずれかの有効期限付きケーパビリティの有効期限が切れ，*ReqM* もしくは *Dt* が移動できなくなってしまうと，初期の式から 10 単位時間

経過後に $ReqM$ を再送することができる．10 単位時間経過（$t < 10$ の t を経過時間とする）までに Dt を受け取ると以下のように $open\ x$ が活性化される．

$$
\begin{array}{l}
Clnt[\\
\quad Dt[\,] \mid open\ x \mid x[\,] \\
\qquad \mid wait(10-t).open\ x.REQM \\
] \mid Srvr[\\
\quad !open\ arrvM.Dt[out\ Srvr.in(5)\ Clnt.arrvD[out\ Dt]] \\
]
\end{array}
$$

この $open\ x$ が消費されると，$REQM$ を先行している $open\ x$ は活性化したとしても消費できないためメッセージは再送されない．

3. 時間に依存した振舞いの記述

HTAC を用いて，時間に依存した振舞いを含む動的な階層構造を持つ系を記述することを考える．ここでは，典型的な時間に依存した振舞いについて，それらを記述する方法を述べる．

3.1 タイムアウト動作の記述

タイムアウト動作を表現することにより，時間に依存した振舞いを柔軟に表現可能となる．

指定した時間 t 以内に記述したイベントを実行した場合は P として振る舞い，実行しなかった（タイムアウトした）場合 Q として振る舞うような記述を，HTAC では以下のように表現する．このとき，タイムアウト監視されるケーパビリティを「$in(t)\ L$」とする．

$$K[(\nu\ x)(in(t)\ L.open\ x.P \mid wait(t).open\ x.Q \mid x[\,])] \mid L[\,]$$

x アンビアントは，その後の振舞い P と Q を排他的に選択するタイムアウト記述のために導入した．x は P と Q の中には現れない制御アンビアントであり，他に影響を及ぼさないよう名前 x の有効範囲に制限をかけている．$(t-t')$ 単位時間経過後（$t' < t$）に「K Enter L」が実行されると

$$
\begin{array}{ll}
\xrightarrow[(t-t')]{\tau} & K[(\nu\ x)(in(t')\ L.open\ x.P \mid \\
 & \quad wait(t').open\ x.Q \mid x[\,])] \mid L[\,] \\
\xrightarrow{\delta} & L[K[(\nu\ x)(open\ x.P \mid wait(t').open\ x.Q \mid x[\,])]] \\
\rightarrow & L[K[(\nu\ x)(P \mid wait(t').open\ x.Q)]]
\end{array}
$$

と遷移し，その後 t' 単位時間経過し

$$\xrightarrow[(t')]{\tau} \quad L[K[(\nu\ x)[(P \mid open\ x.Q)]]$$

となっても，すでに x という制御アンビアントは消滅しているため $open\ x$ は消費可能でないので，$open\ x.Q$ はその後の遷移に影響を与えず，K は $K[P]$ として振る舞う．一方，t 単位時間経過以内に「K Enter L」が実行されなかった（タイムアウトした）場合

$$
\begin{array}{ll}
\xrightarrow[(t)]{\tau} & K[(\nu\ x)(0 \mid open\ x.Q \mid x[\,])] \mid L[\,] \\
\rightarrow & K[0 \mid Q] \mid L[\,]
\end{array}
$$

と遷移し，K は以降 $K[Q]$ として振る舞う．以下では

$$(\nu\ x)(M(t).open\ x.P \mid wait(t).open\ x.Q \mid x[\,])$$

をマクロ記法を用いて

$$M(t).\{P\ \texttt{timeout}\ Q\}$$

と記述する．ここで $M(t)$ はタイムアウト監視される有効期限付きケーパビリティである．この記法を用いて，たとえば 5 単位時間以内に B に入ろうとし，入れない場合には C に入るという行動をとる A を次のように記述できる．

$$A[in(5)\ B.\{0\ \texttt{timeout}\ in\ C.0\}]$$

3.2 イベント発生間隔の上下限の指定

タイムアウト記述と $alarm$ アンビアントを用いてアンビアント K 中の有効期限 t 以内に必ず実行すべきイベント K Enter L を以下のような形で記述することができる．

$$K[in(t)\ L.\{P\ \texttt{timeout}\ alarm[\,]\}]$$

この式は t 単位時間経過（タイムアウト）すると $alarm$ アンビアントが活性化し，遷移後の式は終末的となるので，そのような場合をプロセス式の指定するイベント系列から除外している，すなわちタイムアウトしてはならないことを指定していることになる．以下では $alarm$ アンビアントは上記のような形での使用に限定し，有効期限付きケーパビリティ $M(t)$ に対して

$$M(t).\{P\ \texttt{timeout}\ alarm[\,]\}$$

をマクロ記法を用いて

$$M\langle t\rangle.P$$

と記述する．ただし，このような記述をする場合，$M(t)$ が活性となったとき消費可能であるべきである．たとえば次の式では，

$$L[K[in\langle t\rangle\ A.P]] \mid A[\,]$$

A アンビアントが K アンビアントと並列になることはないため，$in\langle t\rangle\ A$ は活性であるが消費できない．その結果，このケーパビリティが消費できないまま時間経過し続け，必ず $alarm$ アンビアントが活性化してしまい終末的プロセスに至る．そのようなイベント系列は排除されるため，この式が指定するイベント系列の集合に t 時間以上経過するイベント系列は含まれない．つまりこの式は t 時間以上経

過してはならないことを意味することになり，適切なものとはいえない．したがってこのマクロ記法を用いたプロセスを定義する場合，活性なケーパビリティが必ず消費可能となることを確認する必要がある．そのためには，プロセス式の遷移系列を網羅的に解析する可達性解析を行う必要がある．

さらに，これらの記法を用いてイベントの発生間隔の上下限を指定できる．たとえば，A が 30 単位時間以上，40 単位時間未満に B に入る記述を以下に示す．

$$A[wait(30).in\langle 10\rangle\ B]$$

また，これらの記法を用いて連続した 2 つのイベント間の発生間隔の上限と下限を指定できる．たとえば，B を出て，30 単位時間以上 40 単位時間未満に C に入る A は次のように記述できる．

$$A[out(\infty)\ B.wait(30).in\langle 10\rangle\ C]$$

このような式については次のようなマクロ記法を用いることにする．

$$A[out(\infty)\ B.in\langle 30..40\rangle\ C]$$

alarm アンビアントは本節で述べたようなイベントの発生間隔の上限を指定するための使用法に限定することとする．

3.3　異なるアンビアントのイベント発生間隔の指定

上記のように，同一アンビアントのイベントの発生間隔の指定はアンビアント内の一連のケーパビリティで指定することが可能であるが，発生間隔を指定する 2 つのイベントが異なるアンビアントに存在する場合，制御アンビアントによるイベント発生情報の通知が必要である．たとえば，A が B を出てから 30 単位時間以上 40 単位時間未満に，C が B から出て行くような記述がしたい場合，以下のように記述すればよい．

$$\begin{array}{l} B[(\nu\ x)(\\ \quad A[out(t_1)\ B.x[out\ A.in\ B.in\ C]]\ | \\ \quad C[open\ x.out\langle 30..40\rangle\ B]) \\ ] \end{array}$$

イベントが発生すると x という制御アンビアントが活性化し，他方のアンビアントに移動すると

$$B[(\nu\ x)(C[x[\ ]\ |\ open\ x.out\langle 30..40\rangle\ B])]\ |\ A[\ ]$$

となり，「x disappear」が起こることによってイベント発生間隔の上下限を指定する．これによって異なるアンビアントのイベント発生間隔を指定することができる．

3.4　遠隔イベント発生間隔の上下限の指定

発生間隔を指定すべき 2 つのイベントが連続したものでないような場合，たとえば B，C の外にいる A が B を経由して C へ向かう際，B 到着時刻と C 到着時刻の間隔の下限と上限を指定したい場合はもう少し複雑な記述が必要になる．本節では，下限を指定する方法と上限を指定する方法について説明する．なお間隔を指定しない場合は，

$$A[\ in(\infty)\ B.out(\infty)\ B.in(\infty)\ C\ ]\ |\ B[\ ]\ |\ C[\ ]$$

と書けばよい．

3.4.1　遠隔イベント発生間隔の下限の指定

B 到着から C 到着までの所要時間の下限 t_1 を指定する場合，B に到着したときに下限時間を計測するタイマ s を起動し，待機時間満了により，タイマ s 内の待機ケーパビリティが消費されたことを確認してから C に到着することを指定する場合，次のような記述をすればよい．

$$\begin{array}{l} A[in(\infty)\ B.(\nu\ s,x)(\\ \qquad out(\infty)\ B.open\ x.in(\infty)\ C \\ \quad |\ s[wait(t_1).x[out\ s]])]\ |\ B[\ ]\ |\ C[\ ] \end{array}$$

このプロセス式では，s アンビアントはケーパビリティ $in(\infty)\ B$ に先行されているので活性ではない．ここで「A Enter B」を実行すると

$$\begin{array}{ll} \xrightarrow{\delta} & B[A[(\nu\ s,x)(\\ & \qquad out(\infty)\ B.open\ x.in(\infty)\ C \\ & \quad |\ s[wait(t_1).x[out\ s]])]] \\ & |\ C[\ ] \end{array}$$

に遷移し，s アンビアントと s 中の $wait(t_1)$ ケーパビリティが活性化する．その後 $(t_1 - t_1')$ 単位時間経過した後に「A Exit B」実行した場合

$$\begin{array}{ll} \xrightarrow[(t_1-t_1')]{\tau} & B[A[(\nu\ s,x)(\\ & \qquad out(\infty)\ B.open\ x.in(\infty)\ C \\ & \quad |\ s[wait(t_1').x[out\ s]])]] \\ & |\ C[\ ] \\ \xrightarrow{\delta} & A[(\nu\ s,x)(open\ x.in(\infty)\ C \\ & \quad |\ s[wait(t_1').x[out\ s]])]\ |\ B[\ ]\ |\ C[\ ] \\ \xrightarrow[(t_1')]{\tau} & A[(\nu\ s,x)(open\ x.in(\infty)\ C \\ & \quad |\ s[x[out\ s]])]\ |\ B[\ ]\ |\ C[\ ] \\ \rightarrow & A[(\nu\ s,x)(open\ x.in(\infty)\ C\ |\ x[\ ]\ |\ s[\ ])] \\ & |\ B[\ ]\ |\ C[\ ] \\ \rightarrow & A[(\nu\ s)(in(\infty)\ C\ |\ s[\ ])]\ |\ B[\ ]\ |\ C[\ ] \end{array}$$

と遷移し，「A Enter B」が実行されてから t_1 単位時間経過後はじめて $in(\infty)\ C$ が活性化する．

$$M_1.(\nu\ s,x)(N.open\ x.M_2P\ |\ s[wait(t_1).x[out\ s]])$$

をマクロ記法を用いて

$$M_1.(N \mid \langle t_1..\rangle).M_2.P$$

と記述することにする．これにより M_1 と M_2 との発生間隔の下限を指定できる．このとき，N はケーパビリティ列である．

3.4.2　遠隔イベント発生間隔の上限の指定

B 到着から C 到着までの所要時間の下限 t_1 を指定したプロセス式

$$A[\ in(\infty)\ B.(out(\infty)\ B \mid \langle t_1..\rangle).in(\infty)\ C] \mid B[\,] \mid C[\,]$$

中の「$out(\infty)\ B.in(\infty)\ C$」の部分を，所要時間の上限 t_2 を指定したものに書き換えることを考える．仮にそのような記述を

$$(out(\infty)\ B.in(\infty)\ C \mid \langle..t_2\rangle)$$

と表すとする．

まず，$out(\infty)\ B$ が活性化してから 1 単位時間経過する以前に「A Exit B」が実行されたとすると，以降の振舞いは $in\langle t_2\rangle\ C$ となる．一方，「A Exit B」が実行されることなく 1 単位時間が経過したとすると以降の振舞いは $(out(\infty)\ B.in(\infty)\ C \mid \langle..t_2-1\rangle)$ となる．すなわち上記の式はタイムアウト記述を用いて

$$\begin{array}{l} out(1)\ B.\{in\langle t_2\rangle\ C \\ \quad \texttt{timeout}\ \ (out(\infty)\ B.in(\infty)\ C \mid \langle..t_2-1\rangle)\} \end{array}$$

と表現できる．同様の展開を続けると以下のような式が得られる．

$$\begin{array}{l} out(1)\ B.\{in\langle t_2\rangle\ C \\ \quad \texttt{timeout}\ \ out(1)\ B.\{in\langle t_2-1\rangle\ C \\ \quad \texttt{timeout}\ \ out(1)\ B.\{in\langle t_2-2\rangle\ C \\ \ldots \\ \quad \texttt{timeout}\ \ (out(\infty)\ B.in(\infty)\ C \mid \langle..1\rangle)\}\ldots\} \end{array}$$

$(out(\infty)\ B.in(\infty)\ C \mid \langle..1\rangle)$ は 1 単位時間経過するまでに「A Exit B」と「A Enter C」を実行しなければならないので $out\langle 1\rangle\ B.in\langle 1\rangle\ C$ と書け，

$$(out(\infty)\ B.in(\infty)\ C \mid \langle..t_2\rangle)$$

は

$$\begin{array}{l} out(1)\ B.\{in\langle t_2\rangle\ C \\ \quad \texttt{timeout}\ \ out(1)\ B.\{in\langle t_2-1\rangle\ C \\ \quad \texttt{timeout}\ \ out(1)\ B.\{in\langle t_2-2\rangle\ C \\ \ldots \\ \quad \texttt{timeout}\ \ out\langle 1\rangle\ B.in\langle 1\rangle\ C\}\ldots\} \end{array}$$

と表すことができる．

下限指定のマクロ記法とあわせて，下限上限を指定するマクロ記法として

$$M.(N \mid \langle t_1..t_2\rangle).P$$

を用いることとする．ここで N はケーパビリティ列である．またこの遠隔イベント発生間隔の上限の表現方法はケーパビリティ列 N の長さに対し組合せ爆発が起こるが，現実的な記述では問題とならない．

4.　HTAC による物流計画記述

4.1　物流

本論文では，物流とは，階層構造をなすことができるトラックや配送所などの複数のオブジェクトからなる系であると見なす．物流では，オブジェクトの階層構造を変化させるアトミックなオブジェクトの移動（以下，イベントと呼ぶ）が離散時間的に発生し，それらの一連のイベントにより，貨物の目的地への移動などの所期の目的を達する．

物流計画とはこれらのイベント系列の集合を指定することである．時間の概念を考慮する場合，イベント間に時間経過が挿入される．

4.2　ケーパビリティの使い分け

HTAC を用いた物流計画記述では，2 章で述べたように，人や物などのオブジェクトをアンビアントで表現することで階層構造を表現する．そこで，オブジェクト間のイベント発生情報の通知などを表現するための制御アンビアントに対して，人や物などのオブジェクトを表すアンビアントを本論文では物理アンビアントと呼ぶ．以降では，物理アンビアントの頭文字は大文字，制御アンビアントの頭文字は小文字で表現することでこれらを区別する．

物理アンビアントの階層構造（以降，物理階層と呼ぶ）によって系の現在の状態を表し，ケーパビリティや制御アンビアントによってイベント系列を指定する．プロセス式から物理階層を抽出する関数 $\mathcal{H}$ を以下のように定義する．

定義 4.1（物理階層の抽出）

$\mathcal{H}((\nu n)P) = \mathcal{H}(P)$	*restriction*
$\mathcal{H}(0) = 0$	*inactivity*
$\mathcal{H}(P \mid Q) = \mathcal{H}(P) \mid \mathcal{H}(Q)$	*composition*
$\mathcal{H}(!P) = 0$	*replication*
$\mathcal{H}(N[P]) = N[\mathcal{H}(P)]$	*physical ambient*
$\mathcal{H}(n[P]) = \mathcal{H}(P)$	*control ambient*
$\mathcal{H}(trashbox[0]) = \mathcal{H}(0)$	*trashbox ambient*
$\mathcal{H}(M.P) = 0$	*action*

□

$!P$ のプロセス P 内に物理アンビアントが存在することは，同じオブジェクトが無限に存在することを表すことになるため，$!P$ のプロセス P 内に活性な物理アンビアントの存在を禁止する．物流における階層構造，たとえば

$Taxi[Man[P] \mid Q] \mid R$ はタクシーに人が乗っていることを表す．P，Q，R によってオブジェクトのイベント系列を指定する．このときイベントの発生情報の通知のために P，Q，R 間で制御アンビアントのやりとりが必要である．

実際の物流では，貨物の積み込みや積み下ろしといったオブジェクト間の出入りは離散的に発生する．このことから，オブジェクト間の出入りを表現するのに有効期限付きケーパビリティを用いる．たとえば，タクシーに人が乗り込むイベントは

$$Taxi[\,] \mid Man[in(t)\ Taxi] \xrightarrow{\delta} Taxi[Man[\,]]$$

と遷移することで表現する．t 単位時間までに乗り込むなど，イベントの発生時間帯を指定したい場合は，3 章で述べた上限などで指定する．一方，イベントの発生を通知するためなどの制御遷移は瞬時に行われる必要があるため，通常のケーパビリティを用いる．たとえば，タクシーが場所 A に到達した後に人がタクシーから降りるイベント系列を指定したい場合は

$$\begin{aligned}
&Taxi[in\langle t_1\rangle\ A.arrv[in\ Man] \\
&\quad \mid Man[open\ arrv.out\langle t_2\rangle\ Taxi]] \\
&\mid A[\,] \\
&\xrightarrow{\delta} A[Taxi[arrv[in\ Man] \\
&\qquad \mid Man[open\ arrv.out\langle t_2\rangle\ Taxi]]] \\
&\rightarrow A[Taxi[Man[arrv[\,] \mid open\ arrv.out\langle t_2\rangle\ Taxi]]] \\
&\rightarrow A[Taxi[Man[out\langle t_2\rangle\ Taxi]]]
\end{aligned}$$

と記述することで「*Taxi* Enter *A*」「*Man* Exit *Taxi*」というイベント系列を指定できる．制御遷移は *Taxi* が *A* についた瞬間に *Man* が *Taxi* から降りられるように瞬時的に行う．以降，有効期限付きケーパビリティの消費によるプロセス式の遷移を「イベント遷移」と呼ぶことにする．また，時間経過の上限だけでなく下限やタイムアウト動作を表現する場合は，3 章の記述方法によって指定する．

4.3　アンビアント名の拡張

物流計画記述では，同じ役割を果たす物理アンビアントが複数存在（複数のタクシーなど）することがあり，同じアンビアント名を付けるのが適切である．ただし，これらは異なるオブジェクトとして区別する必要があるため，下添え字を付ける．下添え字は任意の文字列で表現し，下添え字を持つアンビアントの遷移規則は以下のようになる．

$$M[in(t_\infty)\ N] \mid N_n[P] \xrightarrow{\delta} N_n[P \mid M[\,]] \quad (Subscript)$$

この遷移規則により，下添え字のみ異なるアンビアントはケーパビリティの対象として区別しない．たとえば，

$$Man[in(t_\infty)\ Taxi] \mid Taxi_1[\,] \mid Taxi_2[\,]$$

はどちらのタクシーにも乗ることができる記述となる．

一方，物流計画記述において，物理アンビアントは状態が変化（タクシーにおける，賃走や空車など）することがある．アンビアントの状態を表現する場合，名前の一部として上添え字をつける．上添え字は任意の文字列で表現し，上添え字を持つアンビアントの遷移規則は以下のようになる．

$$M[in(t_\infty)\ N^s] \mid N^s[P] \xrightarrow{\delta} N^s[P \mid M[\,]] \quad (Superscript)$$

この遷移規則により，上添え字のみ異なるアンビアントはケーパビリティの対象として区別する．たとえば，

$$Man[in(t_\infty)\ Taxi^{empty}] \mid Taxi_1^{empty}[\,] \mid Taxi_2^{full}[\,]$$

は空車のタクシー（$Taxi_1^{empty}$）にのみ乗ることができる記述となる．

以下に空車から賃走などへの状態の変化を表すような，N^s を $N^{s'}$ に名前を変更する記述式を示す．これは文献 [4] の *renaming* と同様の記述方法である．

$$\begin{aligned}
&N_1^s[Q \mid in\ N^{s'} \mid N_1^{s'}[out\ N^s.open\ N^s.P]] \\
&\rightarrow N_1^s[Q \mid in\ N^{s'}] \mid N_1^{s'}[open\ N.P] \\
&\rightarrow N_1^{s'}[open\ N^s.P \mid N_1^s[Q]] \\
&\rightarrow N_1^{s'}[P \mid Q]
\end{aligned}$$

この記述では，一時的に状態が変化する前と後の物理アンビアントが混在する．これらの移動については，系の動作ではなく名前変更のための動作であるため，制御遷移によって移動する．このような式については次のようなマクロ記法を用いる．

$$N_1^s[Q \mid (N_1^s\ be\ N_1^{s'}).P]$$

オブジェクトの状態が異なったとしても同じ物理オブジェクトを表現したものであるため，物理階層を抽出する際は上添え字は無視される．

4.4　物流計画と見なせるための条件

HTAC のプロセス式が，物流計画を記述していると見なせるための条件を以下で与える．

定義 4.2（物流的遷移）　$\mathbf{F} \xrightarrow{\delta} G \xrightarrow{*} \mathbf{G}$ が以下を満たすとき，これを物流的遷移という．

(1)「$\xrightarrow{\delta}$」遷移は物理アンビアント N，M に関する「N *Enter* M」，「N *Exit* M」，「M *Enter* N」，「M *Exit* N」のみである．

(2) $G \rightarrow G' \rightarrow G'' \rightarrow \ldots \rightarrow G^{(n)} \rightarrow \ldots$ となる無限遷移列が存在しない．

(3) $G \xrightarrow{*} \mathbf{G}'$ かつ $\mathbf{G} \not\equiv \mathbf{G}'$ なる $\mathbf{G}'$ が存在しない．

(4) $\mathcal{H}(\mathbf{F}, e) \equiv \mathcal{H}(\mathbf{G})$ である（e は $\mathbf{F} \xrightarrow{\delta} G$ で発生するイベント）．　□

ここで $\mathcal{H}(\mathbf{F}, e)$ とは，プロセス式 $\mathbf{F}$ の物理階層にイベ

ント e を適用した後の物理階層であり，以下のように定義する．

定義 4.3（イベント後の物理階層）

$$\mathcal{H}(N[in(t_\infty)\ M.P \mid Q] \mid M[R], N \text{ Enter } M) \equiv \mathcal{H}(M[N[\mathcal{H}(Q)] \mid \mathcal{H}(R)])$$
$$\mathcal{H}(M[N[out(t_\infty)\ M.P \mid Q] \mid R], N \text{ Exit } M) \equiv \mathcal{H}(N[\mathcal{H}(Q)] \mid M[\mathcal{H}(R)])$$

□

これは，$\mathbf{F}$ が $N[in(t_\infty)\ M.P \mid Q] \mid M[R]$ のとき，$\mathbf{F} \xrightarrow{\delta} G$ で発生したイベントが「N Enter M」であれば，$\mathcal{H}(\mathbf{F}, e)$ は $\mathbf{F}$ の物理階層 $\mathcal{H}(\mathbf{F})$ に「N Enter M」が発生した物理階層 $M[N[\mathcal{H}(Q)] \mid \mathcal{H}(R)]$ であることを表している．

これらの条件が必要な理由を以下に示す．

条件 (1)　4.2 節より「$\xrightarrow{\delta}$」遷移はオブジェクト間の出入りの表現に用いられるため，「$\xrightarrow{\delta}$」遷移は物理アンビアント間の遷移のみに限定する．また，オブジェクトの消滅は物流として表現する必要がないため，$open\ (t_\infty)M$ は用いない．

条件 (2)　HTAC による物流計画において，一連の制御遷移が終わらず，時間経過ができないことは計画として不適切である．無限遷移列を持つ例を以下に示す．

$$\begin{aligned}
&A[\,] \mid C[\ in(5)A] \mid !b[in\ A]\\
&\to A[b[\,]] \mid C[\ in(5)A] \mid !b[in\ A]\\
&\to A[b[\,] \mid b[\,]] \mid C[\ in(5)A] \mid !b[in\ A]\\
&\to A[b[\,] \mid b[\,] \mid b[\,]] \mid C[\ in(5)A] \mid !b[in\ A]\\
&\to \cdots
\end{aligned}$$

これは (repl) によって $b[in\ A]$ が複製され，つねに $in\ A$ が消費可能となり，「b enter A」が発生し続けてしまい，一連の制御遷移が終わらない．

条件 (3)　制御遷移の順序の違いによって非決定的になるのは適切でない．1 イベントにつき到達する安定な式はただ 1 つである．

条件 (4)　一連の制御遷移によって物理階層が変動することは不適切である．$\mathcal{H}(G)$ は名前変更などによって同じオブジェクトを表現した物理アンビアントが複数存在することがあるため，必ずしも $\mathcal{H}(\mathbf{F}, e) \equiv \mathcal{H}(G)$ とはならない．よって式 $\mathbf{F}$ の物理階層にイベント e を適用したものと得られた安定な式 $\mathbf{G}$ の物理階層は構造合同でなければならない．一連の制御遷移によって物理階層が変動する例を以下に示す．

$$\begin{aligned}
&A[\,] \mid Taxi[in(t_1)\ A.arrv[in\ Man.out\ Taxi]\\
&\quad \mid Man[open\ arrv.out(t_2)\ Taxi]]\\
&\xrightarrow{\delta} A[\\
&\qquad Taxi[arrv[in\ Man.out\ Taxi]\\
&\qquad \mid Man[open\ arrv.out(t_2)\ Taxi]]\\
&\quad]\\
&\to A[Taxi[\\
&\qquad \mid Man[arrv[out\ Taxi] \mid open\ arrv.out(t_2)\ Taxi]]\\
&\quad]\\
&\to A[Taxi[\\
&\qquad \mid Man[out\ Taxi] \mid out(t_2)\ Taxi]]\\
&\quad]\\
&\to A[Taxi[\,] \mid Man[out(t_2)\ Taxi]]
\end{aligned}$$

定義 4.4（物流的時間遷移）　$\mathbf{F} \xrightarrow{\tau} G \xrightarrow{*} \mathbf{G}$ が以下を満たすとき，これを物流的時間遷移という．

(i)　$G \to G' \to G'' \to \ldots \to G^{(n)} \to \ldots$ となる無限遷移列が存在しない．

(ii)　$G \xrightarrow{*} \mathbf{G}'$ かつ $\mathbf{G} \not\equiv \mathbf{G}'$ なる $\mathbf{G}'$ が存在しない．

(iii)　$\mathcal{H}(\mathbf{F}) \equiv \mathcal{H}(\mathbf{G})$ である．　□

(i)，(ii) は定義 4.2 の (2)，(3) と同様である．(iii) は時間経過によって物理階層が変動してはならないことを示している．

定義 4.5（**1 ステップ物流的プロセス式**）　与えられたプロセス式 $\mathbf{F}$ からのすべての遷移が物流的遷移，または，物流的時間遷移であるとき，式 $\mathbf{F}$ は 1 ステップ物流的プロセス式であるという．　□

定義 4.6（**物流的プロセス式**）　与えられたプロセス式 $\mathbf{F}$ からの到達可能な安定な式 $\mathbf{F}'$ がすべて 1 ステップ物流的プロセス式であれば，式 $\mathbf{F}$ は物流的プロセス式であるという．　□

4.5　*trashbox* アンビアントを用いた物流的プロセス式

π 計算 [12] などで排他的選択のために用いられていた「+」演算子は，AC や AC を拡張したプロセス代数ではその他の構文を利用することによって模倣できることが示されている [4], [9]．HTAC にも「+」演算子は存在しないが，これらの模倣法を用いると定義 4.2 に反してしまうため排他的選択の表現が容易にはできない．HTAC では，部分式を破棄するために特別なアンビアント *trashbox* に関わる制約を遷移規則に含めている．*trashbox* アンビアントを用いることで定義 4.2 を満たした排他的選択動作が表現できる例を以下に示す．

$$\begin{aligned}
&F = Man[\\
&\quad in\langle 5\rangle\ CarB.Man^{pre}[out\ Man\\
&\qquad .check[in\ Man.out\ Man.ok[out\ check]]\\
&\qquad \mid open\ ok.open\ check.(Man^{pre}\ be\ Man)\\
&\qquad .(y[in\ Man] \mid P) \mid trashbox[\,]]\\
&\quad \mid in\langle 5\rangle\ CarC.Man^{pre}[out\ Man\\
&\qquad .check[in\ Man.out\ Man.ok[out\ check]]\\
&\qquad \mid open\ ok.open\ check.(Man^{pre}\ be\ Man)\\
&\qquad .(y[in\ Man] \mid Q) \mid trashbox[\,]]\\
&\quad \mid in\ Man^{pre}.open\ y.in\ trashbox\\
&] \mid CarB[\,] \mid CarC[\,]
\end{aligned}$$

このプロセス式は Man が 5 単位時間以内に $CarB$ に入れば P として振る舞い，$CarC$ に入れば Q として振る舞う．どちらの遷移も起こらずに 5 単位時間経過すると定義 2.8 の $alarm$ アンビアントが活性化してしまい，終末的プロセスとなる．よってこれは 5 単位時間経過するまでに「Man Enter $CarB$」または「Man Enter $CarC$」の遷移が起こらなければならないというプロセス式である．簡単のため $in\langle 5\rangle\ CarC$ に続く部分式を PROC とする．

この F は $F \to Q$ なる Q を持たないため，以下では安定な式 $\mathbf{F}$ と記す．$\mathbf{F}$ から可能な「$\stackrel{\delta}{\to}$」遷移は「Man Enter $CarB$」もしくは「Man Enter $CarC$」のみであるため，定義 4.2 の条件 (1) を満たしている．「Man Enter $CarB$」が起こると一連の制御遷移後，得られる安定な式 $\mathbf{G}$ は

$$CarB[Man[\ P \mid trashbox[Man[in\langle 5\rangle\ CarC.PROC]]\ ]] \mid CarC[\ ]$$

となる．無限遷移列を持たないため，定義 4.2 の条件 (2) を満たしている．制御遷移の順序によって非決定的にならないため，定義 4.2 の条件 (3) を満たしている．$\mathcal{H}(\mathbf{F}, Man$ Enter $CarB)$ は

$$CarB[Man[\ ]] \mid CarC[\ ]$$

であり $\mathcal{H}(\mathbf{G})$（P 内に物理アンビアントは存在しないものとする）と構造合同であるため，定義 4.2 の条件 (4) を満たしている．

一方，式 $\mathbf{F}$ から「Man Enter $CarC$」が起こった場合，

$$CarC[Man[\ Q \mid trashbox[Man[in\langle 5\rangle\ CarC.PROC]]\ ]] \mid CarB[\ ]$$

が得られる．

ここで $trashbox$ アンビアントがなければ，$\mathbf{G}$ 内の $in\langle 5\rangle\ CarC$ が時間切れとなり終末的プロセスになってしまうことから，「Man Enter $CarB$」を含むイベント系列は排除されてしまう．$CarC$ が選ばれる場合も同様であり，結局排他的選択動作が記述できないことになってしまう．

5. 処理系を用いた記述実験

5.1 物流計画と見なせる HTAC 式の処理系

物流計画と見なせる HTAC プロセス式を遷移させるための処理系を開発した．与えられた安定なプロセス式から HTAC の遷移規則どおりに消費可能な有効期限付きケーパビリティを列挙し，ユーザが消費するケーパビリティや時間経過を選択できる．また非安定な式に遷移した場合，定義 4.2 の条件 (3) によって「→」遷移の順序による到達する安定な式に違いがあってはならないため，合流性を満たしていることを検出した後に，一気に安定な式まで遷移させる．しかし，定義 4.2 の条件 (2) が満たされない場合の検出機能はない．**図 1** に実行画面を示す．

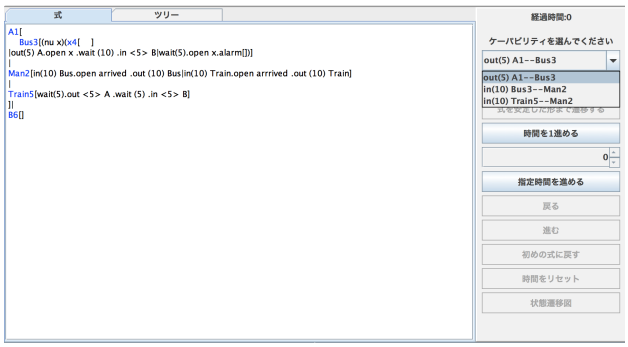


図 1　処理系の実行画面
Fig. 1　GUI of the processing system for HTAC.

右側上部のプルダウンメニューによって現在のプロセス式から消費可能なケーパビリティを選択できる．遷移を 1 つ前のプロセス式に戻したり，初期の式に戻したりするバックトラック機能やマクロの展開機能を持つ．また，定義 4.2，4.4，4.5 の一部を満たしているか随時チェックしている．

これによってプロセス式が意図どおりの動作をするかを確認することができる．処理系は Java 言語によるもので，規模はクラス数 32，総行数 18,000 である．

5.2 記述実験

HTAC による物流計画の記述実験を行った．以下に記述した系の概要を示す．

BusOrTaxi

これは，旅行者が鉄道駅からシャトルバスまたはタクシーで空港へ移動し，飛行機に乗り換えて目的地へ移動する 7 個のオブジェクトからなる系である．シャトルバスは空港鉄道駅間を往復するようプロセス定義式を用いて再帰的に定義しており，鉄道駅空港間の所要時間の下限と上限，それぞれにおける停車時間を指定している．タクシーは旅行者が乗るまで駅で待機し，旅行者が乗るとバスより短い所要時間で駅から空港へ向かうよう，飛行機は旅行者が乗ったことを確認することなく定刻に出発するよう記述している．旅行者は一定時間までバスで空港へ移動しようとするが，それを過ぎるとタクシーで移動するよう記述し，旅行者が飛行機に間に合うような記述になっている．

1 単位時間 = 1 秒と想定しており，飛行機は初期の式から 90 分経過すると出発する．よって旅行者は 50 分経過するまでにバスに乗らなかった場合，5 分以内にタクシーに乗り，15 分以内に飛行機に乗り込めるよう時間設定してある．バスに乗車するイベント系列で順当に遷移させていくと，所要時間は 150 分程度である．

GotoUniv

これは，学生が電車と徒歩で移動し，講義開始時刻までに教室に入室すると出席確認証を受け取る 8 個のオブジェクトからなる系である．電車が一定時間以上遅れた際には駅で遅延証明書が発行され，遅延証明書を提出すれば講義開始時刻以降でも出席確認証を受け取ることができる．一方，電車が遅延した場合でも，遅延証明書を受け取らずに教室に入った学生は出席確認証を受け取ることはできない．

1 単位時間 = 1 秒と想定しており，初期の式から 20 分後に講義は開始し，16 分 40 秒経過から 20 分経過までの間に学生が教室内にいれば出席確認証を受け取ることができる．電車は初期の式から 10 分経過すると最寄駅に到着できるようになるが，10 分経過からさらに 11 分 40 秒経過すると電車が遅延したものと見なし，遅延証明書を受け取ることができるようになる．また初期の式から電車が 60 分経過するまでに最寄駅に到着しなかった場合，電車は止まっているものと見なし，大学は休講となる．電車の遅延もなく順当に遷移させていくと，所要時間は 20 分で学生が出席確認証を持ち，講義は開始される．

PushBtnSgnl

これは，荷物を持った 2 人の歩行者と 2 台の車が押しボタン式信号で管理された交差点を安全に通行し，歩行者は向こう岸に着くと荷物を降ろすような 9 個のオブジェクトからなる系である．初期の式では車が交差点に入ることができ，歩行者は交差点に入れない．歩行者がボタンを押すと時間（下限と上限を指定）経過後，自動車が交差点に入れなくなり，さらにそのあと一定時間経過後，歩行者が一定時間交差点に入れるようになる．一方，自動車が交差点に入れる時間については下限のみが設定され，歩行者が交差点を出て行くまでの十分な時間待機している．さらに自動車と歩行者に対しては，交差点に入ってから出るまでの時間の下限と上限を設定しており，自動車と歩行者が同時に交差点内に存在することがないような記述になっている．この記述では，以上のようなスケジュールを再帰的に定義されたタイマ（制御アンビアント）を用いて管理している．

1 単位時間 = 1 秒と想定しており，初期の式から 30 秒間は必ず車が横断できる時間帯である．歩行者が横断を開始すると 9 秒以内には必ず横断を完了する．車が横断を開始すると 4 秒以内には必ず横断を完了する．また，歩行者は横断開始まで最大 70 秒待機し，車は最大 40 秒待機するような時間設定となっている．

BusOrTaxi のプロセス式

以下に BusOrTaxi の記述を示す．

$$
\begin{array}{l}
Station[\\
\quad Man[\\
\quad\quad in(3000)Bus^{Stop}.\{\\
\quad\quad\quad (x[out\ Man.out\ Bus^{Moving}\\
\quad\quad\quad\quad .in\ Bus^{Moving}.in\ Man.rideBus[out\ x]]\\
\quad\quad\quad\quad |\ open\ rideBus.wait(1200).out\langle 10\rangle\ Bus^{Stop}\\
\quad\quad\quad\quad .wait(600).open\ ok.in\langle 300\rangle\ Plane\\
\quad\quad\quad\quad .open\ arrivedAirPortB.out\langle 120\rangle\ Plane)\\
\quad\quad\quad timeout\\
\quad\quad\quad\quad in\langle 300\rangle\ Taxi.(ride[out\ Man]\\
\quad\quad\quad\quad |\ open\ arrivedAirPortA.out\langle 10\rangle\ Taxi\\
\quad\quad\quad\quad .wait(600).open\ ok.in\langle 300\rangle\ Plane\\
\quad\quad\quad\quad .open\ arrivedAirPortB.out\langle 120\rangle\ Plane)\}\\
\quad]\ |\ Taxi[\\
\quad\quad open\ ride.out\langle 1\rangle\ Station.wait(900)\\
\quad\quad\quad .in\langle 1\rangle\ AirPortA.arrivedAirPortA[in\ Man]\\
\quad]\\
]\ |\ Bus^{Moving}[\\
\quad\quad LOOP\\
]\ |\ AirPortA[\\
\quad Plane[wait(5100).\\
\quad (ok[out\ Plane.in\ Man]\\
\quad |\ wait(300).out\langle 1\rangle\ AirPortA.wait(3600)\\
\quad\quad .in\langle 1\rangle\ AirPortB.arrivedAirPortB[in\ Man])]\\
]\ |\ AirPortB[\ ]\\
LOOP\ \triangleq\\
\quad in\langle 1\rangle\ Station.(Bus^{Moving}\ be\ Bus^{Stop}).wait(300)\\
\quad\quad .(Bus^{Stop}\ be\ Bus^{Moving}).out\langle 1\rangle\ Station\\
\quad\quad .wait(1200).in\langle 1\rangle\ AirPortA\\
\quad\quad .(Bus^{Moving}\ be\ Bus^{Stop}).wait(300)\\
\quad\quad .(Bus^{Stop}\ be\ Bus^{Moving})\\
\quad\quad .out\langle 1\rangle\ AirPortA.wait(1200).LOOP
\end{array}
$$

この記述ではタイムアウトマクロ数は 1，上限マクロ数は 15，名前変更マクロ数は 4 である．プロセス $LOOP$ 内の 1 行目は上限マクロと名前変更マクロを 1 回ずつ用いておりケーパビリティ数 1，アンビアント 0 である．

$$
in\langle 1\rangle\ Station.(Bus^{Moving}\ be\ Bus^{Stop}).wait(300)
$$

マクロを展開すると

$$
\begin{array}{l}
(\nu\ x)(in(1)\ Station.open\ x.(in\ Bus^{Stop}\\
\quad |\ Bus^{Stop}[out\ Bus^{Moving}\\
\quad\quad .open\ Bus^{Moving}.wait(300)])\\
\quad |\ wait(1).open\ x.alarm[\ ]\ |\ x[\ ])
\end{array}
$$

となり，ケーパビリティ数 7，アンビアント数 3 削減できていることが分かる．

その他の記述に関しても，3 章と 4.3 節で説明したマクロを用いて HTAC のプロセス式として記述した（付録 A.1，A.2）．それぞれの記述例のアンビアント数（物理アンビアント数），ケーパビリティ数（有効期限付きケーパビリティ数 + 待機ケーパビリティ数），マクロの使用数を**表 1** 示す．

表 1　記述の規模
Table 1　Size of the formulae.

	ambient 数	capability 数	マクロ数
BusOrTaxi	13(7)	27(8)	20
GotoUniv	19(8)	47(17)	5
PushBtnSgnl	57(9)	125(25)	22

表 2　記述の動作
Table 2　Behaviors of the formulae.

	イベント数	個別イベント数	所要時間
BusOrTaxi	22〜24	4	9,000
GotoUniv	6〜12	4	1,200
PushBtnSgnl	12〜20	5	49

BusOrTaxi と GotoUniv では，オブジェクト数，イベント数に比べてプロセス式の規模は極端に大きくなっていない．BusOrTaxi のプロセス式をマクロ展開した場合，アンビアントの総数は 48，ケーパビリティの総数は 102 となるので，マクロ記法を用いることで簡潔な記述が可能になっていると考えられる．

PushBtnSgnl では，タイマが時間の流れを一元的に把握し，各物理アンビアントを制御するための制御アンビアントを多用するような記述になっているため，規模が増大している．そのようなスタイルの記述に対して有効なマクロの導入が必要であると考えられる．

すべての可能なイベント系列を試してはいないが，処理系を用いて各プロセス式が所期の動作を行っていること，その過程で得られたそれぞれの安定なプロセス式が 1 ステップ物流的であることを確認した．各プロセス式の典型的なイベント系列について，系列全体のイベント数，主たるオブジェクト（旅行者，学生，歩行者）個別のイベント数，所要時間を**表 2** に示す．

BusOrTaxi では，バスが旅行者を空港へ運んだ後も駅空港間を往復しているため，系列全体のイベント数が多くなっている．

6. 物流計画記述に基づく物流監視システム

HTAC の物流的プロセス式による物流計画記述に基づく次のような物流監視システムの構築を考える．(i) 物流監視システムでは現時点の系の階層構造と，以降の物流計画を表現する HTAC の安定なプロセス式を保持する．(ii) 系中の各オブジェクトに IC タグなどを装着し，タグリーダによってすべての移動を検知し，監視システムに通知する．(iii) タイマが単位時間の経過ごとに監視システムに通知する．(iv) オブジェクトの移動，または単位時間経過の通知を受けた監視システムはプロセス式を調べ，通知に対応した遷移が正当な遷移であるかを確認し，プロセス式を次の安定なプロセス式まで遷移させる．

監視システムではオブジェクトの移動通知を受け取ると，現在のプロセス式で当該の「$\xrightarrow{\delta}$」遷移が可能であることを確認し，その遷移を実行し，さらに安定な式が得られるまで「$\rightarrow$」遷移を繰り返す．当該の「$\xrightarrow{\delta}$」遷移が可能でない場合，物流計画に反した移動を検知したとして警告を発する．BusOrTaxi の例では，バスが空港に到着後，人が下車する前に駅へ向かって発車したような場合があり，その場合警告によりバスの発車の中止を運転手に促すことになる．

単位時間経過の通知を受けた場合は「$\xrightarrow{\tau}$」遷移を実行し，安定な式が得られるまで「$\rightarrow$」遷移を繰り返す．「$\xrightarrow{\tau}$」遷移により *alarm* が活性化した場合にも，計画で指定した時間内に必要な移動が起らなかったことを検知したとして警告を発する．これは PushBtnSgnl の例では，故障などにより交差点内で自動車が立ち往生しているような場合に相当し，警告により，人手による何らかの救済を促すことができる．

また，物流監視システムの延長として，誘導システムの構築も可能と考えられる．誘導システムでは物流監視システム同様のプロセス式を保持し，式中に存在する物理アンビアントに可能な「$\xrightarrow{\delta}$」遷移が存在すれば，当該のオブジェクトに対して，その移動を促すようなサインを表示する．逆に可能な「$\xrightarrow{\delta}$」遷移が存在しないオブジェクトに対しては，赤信号によって待機すべきであることを表示する．PushBtnSgnl の例では，誘導システムは歩行者が交差点に入るような遷移が可能であれば青を表示し，可能でない場合に赤を表示する歩行者用信号のような役割を果たす．

7. 関連研究

HTAC と同様に，物の移動の表現に時間的な制約を加えられるよう，AC に対し様々な時間拡張が提案されている [1], [2], [6], [13]．これらの中で，特に文献 [1] で提案されている Timed Mobile Ambient（以下 tMA）は，アンビアントやケーパビリティに有効期限を設け，期限までに実行されなければこれらのアンビアントやケーパビリティが消滅するなど，HTAC に近いものになっている．しかし tMA の遷移規則では，プロセス式内でケーパビリティが消費可能でないことが時間経過の条件となっているため，消費できても消費しないという時間経過に関する非決定的な記述が直接的にはできない．たとえば 5.2 節で示した BusOrTaxi のプロセス式の 3 行目にある $in(3000)Bus^{Stop}$ は，3,000 秒の間に *Man* が *Bus* に乗り込むという動作を表しているが，その間に *Bus* が *Man* の並行な位置に来たとしても，必ずしも *Man* が *Bus* に乗ることはないという事象も表している．つまり HTAC では，*Man* が *Bus* に乗ることが可能な場合でも，乗る，または乗らないという非決定的な動作を $Man[in(3000)Bus^{Stop}\cdots] \mid Bus^{Stop}[\cdots]$ という式で表すことが可能である．

文献 [2] では，アンビアント内で予期しない通信が起きないよう型システムの導入による tMA の拡張が提案されている．またこの中で，タクシーを利用する顧客とタクシーの間の動作を取り上げ，tMA による物流システムのモデル化が例示されている．しかし時間経過に関する構文や遷移規則は文献 [1] と同じであるため，やはり時間経過に関する非決定的な記述が直接的にはできない．

文献 [6] では，文献 [3] と同様にアンビアント名をグループ分けするための型に関する拡張，時間を表す変数 T_i $(i \geq 0)$ をプロセス式の前，および各遷移（→）の前に記述することを許す時間に関する拡張，各アンビアントの状態を表すプロセスに関する拡張，そして，これらの情報を利用した条件付きケーパビリティの導入が提案されている．以下は文献 [6] で示されているプロセス式の一部である．

$$T_0 : \mathit{Message} :: m[\mathit{cin}(\text{条件})\ b] \mid \mathit{Buffer} :: b[0]$$

cin（条件）b が条件付きケーパビリティであり，条件（たとえば b の中のメッセージ数が上限以下）が成立すれば以下のような遷移が発生する．

$$T_1 :\rightarrow \mathit{Buffer} :: b[\mathit{Message} :: m[]]$$

これにより，さらに $T_1 - T_0 < T$（m が b に T 時間未満で移動する）などといった時間に関する条件なども表現できる．また条件の記述には通常の論理記号（$\vee$ や $\wedge$ など）が許されているため，時間に関しても柔軟な制約が記述できることが予想される．しかし，HTAC や tMA のように期限が過ぎれば無効になるといった動作の表現には対応していないため，本論文で示したような物流システムの記述には適していない．

文献 [13] では，モバイルデバイスによるアドホックネットワークを表現し，その性質を検証するためのプロセス代数である Timed Calculus for Mobile Ad Hoc Networks（以下 TCMN）が提案されている．TCMN では，各ノードはアンビアントに似た $n[Q]^c_{l,r}$ のような式で表現され，これらが他のプロセス代数と同様並列演算子 | で複数結合されネットワークが表現される．ここで c，l，r はそれぞれ通信チャネル名，地点名，電波の届く有効範囲の半径を表す．

ただし AC と異なり Q にノードを記述することは許されず，したがって階層構造を表現することはできない．Q として許される $\langle v \rangle^{\delta}$ は出力動作を表し，これは v というメッセージが送信されるのにかかる時間が δ であることを表している．この δ は，その時間内に他の通信が発生した場合衝突が起きたことを検知するために使われるため，HTAC のように移動に関する時間制約を表現することはできない．このように TCMN は，ネットワーク内のデバイス間の通信や通信の衝突などを表現するための言語であるため，物流システムを表現することには適していない．

8. おわりに

本論文では，AC に有効期限付きケーパビリティと待機ケーパビリティを追加したプロセス代数である HTAC を提案し，この拡張により時間に依存した振舞いを表現できることを示した．そして，HTAC を用いたプロセス式が物流計画と見なせるための条件として，物流的プロセス式を定義した．さらに，いくつかの記述実験により，いくつかの種類の物流が HTAC により物流的プロセス式として適切に表現できることを確認した．

しかしながら，HTAC 式が物流的プロセス式であること，イベント発生間隔の上限を指定した場合のイベントの発生可能時間が一定時間以上消費可能であることなど，網羅的な可達性解析が必要であり，その実装と効率化が今後の課題である．これについては，モデル検査系の一部として，現在開発中である．

謝辞　本研究は科研費（25330095，15K00040）の助成を受けたものである．

参考文献

[1] Aman, B. and Ciobann, G.: *Timed Mobile Ambient for Network Protocols*, Vol.5048, Elsevier Science (2008).
[2] Aman, B. and Ciobann, G.: Mobile Ambient with Timers, *Mobility in Process Calculi and Natural Computing*, pp.26–36, Springer (2011).
[3] Cardelli, L., Ghelli, G. and Gordon, A.D.: Types for the Ambient Calculus, *Infomation and Computation 177*, pp.160–194 (2002).
[4] Cardelli, L. and Gordon, A.D.: Mobile Ambients, *Theoretical Computer Science*, Vol.240, pp.177–213 (2000).
[5] Cardelli, L.: Abstractions for Mobile Computation, Lecture Notes in Computer Science, Vol.1603, pp.51–94 (1999).
[6] Coronato, A. and De Pietro, G.: Formal Specification and Verification of Ubiquitous and Pervasive Systems, *ACM Trans. Autonomous and Adaptive Systems*, Vol.6, No.1, pp.9:1–9:6 (2011).
[7] 橋本隆弘，加藤　暢，樋口昌宏：多重 Ambient Calculus と UHF 帯 RFID 機器を用いた海上物流監視システム，情報処理学会論文誌：プログラミング，Vol.6, No.2 (2013).
[8] Hirschkoff, D., Pous, D. and Sangiorgi, D.: An Efficient Abstract Machine for Safe Ambients, *The Journal of Logic and Algebraic Programming*, pp.114–149 (2007).
[9] Kato, T.: An Equational Relation for Ambient Calculus, 情報処理学会論文誌, Vol.46, No.12, pp.3016–3029 (2005).
[10] Kato, T. and Higuchi, M.: A Handling Management System for Freight with the Ambient Calculus and UHF RFID tags, *Proc. 15th International Conference on Network-Based Information Systems*, pp.364–371 (2012).
[11] 森本大輔，加藤　暢，樋口昌宏：Ambient Calculus を用いた物流検査システム，情報処理学会論文誌：プログラミング，Vol.48, No.SIG 10, pp.151–164 (2007).
[12] Parrow, J.: An Introduction to the π-Calculus, *Handbook of Process Algebra*, pp.479–543, North–Holland (2001).
[13] Wang, M. and Lu, Y.: A Timed Calculus for Mobile Ad

Hoc Networks, *Proc. 2nd International Symposium on Computer Science in Russia*, pp.118–134 (2012).
[14] 桑原寛明，結縁祥治，阿草清滋：時間付き π 計算によるリアルタイムオブジェクト指向言語の形式的記述，情報処理学会論文誌，Vol.45, No.6, pp.1498–1507 (2004).

付　　録

A.1 GotoUniv

$$
\begin{array}{l}
Train[\\
\quad Man[\\
\quad\quad in\ isInB.out\ isInB.out(10)\,Train.out(60)B.wait(300)\\
\quad\quad .in(300)\,University.wait(90).in(90)\,Class.y[in\ Ticket]\\
\quad]\ |\\
\quad wait(600).in(\infty)B\ |\\
\quad x[isInB[open\ b.out\ x]]\\
]\ |\ B[\\
\quad b[in\ Train.in\ x.in\ isInB]\ |\\
\quad delay[(\nu\ x)(in\ Train.out\ Train.open\ x\\
\quad\quad\quad |\ wait(700).open\ x.in\ Ticket\ |\ x[\,])]\\
\quad |\ Ticket[open\ delay.in(\infty)Man\\
\quad\quad\quad |\ open\ y.out(\infty)Man.in\langle 10\rangle\ Teacher]\\
\quad |\ trainIsStoped[\\
\quad\quad (\nu\ x)(in\ Train.out\ Train.(open\ x\ |\ ok[\,])\\
\quad\quad |\ wait(3600).open\ x.out\ B.open\ ok.in\ University\\
\quad\quad |\ x[\,])\\
\quad]\\
]\ |\ University[\\
\quad open\ trainIsStoped.(University\ be\ University^{Closed})\\
\quad |\ Class[\ Attendance[\\
\quad\quad\quad wait(1000).open\ tArrived\\
\quad\quad\quad .((\nu\ x)(in(200)Man.open\ x\ |\ wait(200)\\
\quad\quad\quad .open\ x.in\langle 1\rangle\ Teacher.in\ Ticket.out\ Ticket\\
\quad\quad\quad .out(\infty)Teacher.in\langle 1\rangle\ Man\ |\ x[\,]))\\
\quad\quad]\\
\quad]\ |\ Teacher[wait(900).in\langle 100\rangle\ Class\\
\quad\quad .tArrived[out\ Teacher.in\ Attendance]]\\
]
\end{array}
$$

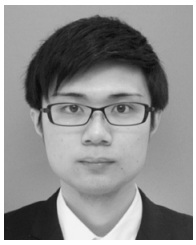
A.2 PushBtnSgnl

$$
\begin{array}{l}
OtherSide[\,]\ |\ Cross[\,]\ |\\
NearCross[\\
\quad manSchedule[\\
TIME\ |\ TIME\\
\quad\quad |\ open\ manApproach.(open\ loop.wait(30)\\
\quad\quad\quad .in\ trashbox\ |\ open\ manApproach\\
\quad\quad\quad |\ manApp[out\ manSchedule.in\ carSchedule]\\
\quad\quad\quad |\ triggerMan[out\ manSchedule.in\ loopTime]\\
\quad\quad\quad |\ triggerCar[out\ manSchedule.in\ CarLoopTime])\\
\quad\quad]\ |\ loopTime[wait(30)\\
\quad\quad\quad .(loop[out\ loopTime.in\ manSchedule]\\
\quad\quad\quad\quad |\ open\ triggerMan.wait(30)\\
\quad\quad\quad\quad .MANSCHEDULE)\\
\quad\quad\quad |\ !open\ loopTime]\ |\\
\quad carSchedule[\\
\quad\quad wait(30).open\ manApp.(manAppA[in\ carGO]\\
\quad\quad\quad |\ manAppB[in\ waitTime]\\
\quad\quad\quad |\ wait(40).in\ trashbox)\ |\\
\quad\quad carGO[open\ manAppA.(carGO\ be\ carStop)]\ |\\
\quad\quad CARTIME\ |\ CARTIME\\
\quad]\ |\ carLoopTime[wait(30)\\
\quad\quad .(carLoop[out\ carLoopTime.in\ carSchedule]\\
\quad\quad\quad |\ open\ triggerCar.wait(40).CARSCHEDULE)\\
\quad\quad |\ !open\ carLoopTime]\\
\quad |\ trashbox[\,]\\
\quad |\ manApproach[open\ requestTime\\
\quad\quad .in\ manSchedule.mango[\,]]\\
\quad |\ manApproach[open\ requestTime\\
\quad\quad .in\ manSchedule.mango[\,]]\\
]\ |\ Car[in(\infty)NearCross.(open\ carTime\\
\quad\quad .(Car\ be\ CarWaitA).open\ carToken\\
\quad\quad .out\langle 2\rangle\ NearCross.in\langle 1\rangle\ Cross.wait(2).out\langle 2\rangle\ Cross\ |\\
\quad\quad requestToken[out\ Car.in\ carSchedule.in\ carTime])]\ |\\
Car[in(\infty)NearCross.(open\ carTime\\
\quad\quad .(Car\ be\ CarWaitB).open\ carToken\\
\quad\quad .out\langle 2\rangle\ NearCross.in\langle 1\rangle\ Cross.wait(2).out\langle 2\rangle\ Cross\ |\\
\quad\quad requestToken[out\ Car.in\ carSchedule.in\ carTime])]\ |\\
Man[\\
\quad\quad in(\infty)NearCross.(open\ time\\
\quad\quad\quad .(Man\ be\ ManWaitA).open\ manToken\\
\quad\quad\quad .out\langle 2\rangle\ NearCross.in\langle 1\rangle\ Cross.wait(6)\\
\quad\quad\quad .out\langle 3\rangle\ Cross.in\langle 1\rangle\ OtherSide.arrv[in\ A]\ |\\
\quad\quad\quad requestTime[out\ Man.in\ manApproach\\
\quad\quad\quad\quad .manName[in\ time]])\\
\quad\quad |\ A[open\ arrv.out\langle 5\rangle\ ManWaitA]\\
\quad]\ |\\
Man[\\
\quad\quad in(\infty)NearCross.(open\ time\\
\quad\quad\quad .(Man\ be\ ManWaitB).open\ manToken\\
\quad\quad\quad .out\langle 2\rangle\ NearCross.in\langle 1\rangle\ Cross.wait(6)\\
\quad\quad\quad .out\langle 3\rangle\ Cross.in\langle 1\rangle\ OtherSide.arrv[in\ B]\ |\\
\quad\quad\quad requestTime[out\ Man.in\ manApproach\\
\quad\quad\quad\quad .manName[in\ time]])\\
\quad\quad |\ B[open\ arrv.out\langle 5\rangle\ ManWaitB]\\
]
\end{array}
$$

```
MANSCHEDULE =
  manSchedule[
    out loopTime.(TIME | TIME |
      open manApproach.(open loop.wait(30)
        .in trashbox | open manApproach
        | manApp[out manSchedule.in carSchedule]
        | triggerMan[out manSchedule.in loopTime]
        | triggerCar[out manSchedule.in CarLoopTime]
      )
  | loopTime[out manSchedule.in loopTime.wait(40)
  .(loop[out loopTime.in manSchedule]
  | open triggerMan.wait(30).MANSCHEDULE)])
  ]
CARSCHEDULE = carSchedule[
    out carLoopTime.(wait(30).open manApp
    .(manAppA[in carGO]
    | manAppB[in waitTime] | wait(40).in trashbox) |
    carGO[open manAppA.(carGO be carStop)] |
    CARTIME | CARTIME |
    carLoopTime[out carSchedule
    .in carLoopTime.wait(30)
    .(carLoop[out carLoopTime.in carSchedule]
    | open triggerCar.wait(40).CARSCHEDULE)])
  ]
TIME = time[
  in mango.out mango.(open manName
    .out manSchedule.in Man |
    waitTime[in assureOK.out assureOK
      .wait(10).manToken[out waitTime]]) |
    assureTime[wait(30)
      .assureOK[out assureTime]]
]
CARTIME = carTime[
  open requestToken.(in carGO.out carGO
    .open x.out carSchedule.in Car.carToken[ ]
    | in carStop.out carStop.open x
      .(carWaitToken[ ] | waitTime[
        wait(40).carToken[out waitTime]]) | x[ ])
  | open carWaitToken.out carSchedule.in Car
]
```

藤坂 吉秀　（正会員）

1992 年生．2015 年近畿大学理工学部情報学科卒業．2017 年同大学大学院博士前期課程修了．現在，（株）アイテック阪急阪神勤務．在学中，プロセス代数の研究に従事．



稲森 啓太　（正会員）

1993 年生．2015 年近畿大学理工学部情報学科卒業．2017 年同大学大学院博士前期課程修了．現在，（株）アイテック阪急阪神勤務．在学中，プロセス代数の研究に従事．

樋口 昌宏　（正会員）

1983 年大阪大学基礎工学部情報工学科卒業．1985 年同大学大学院博士前期課程修了．（株）富士通研究所勤務，大阪大学基礎工学部助手・講師等を経て，現在，近畿大学理工学部情報学科教授．博士（工学）．分散システムの記述，検証，試験に関する研究に従事．

加藤 暢　（正会員）

1997 年岡山大学大学院自然科学研究科博士課程修了．博士（工学）．1998 年日本学術振興会特別研究員（PD）．2000 年より近畿大学理工学部講師．現在，准教授．並行論理型言語の意味論，プロセス代数の研究に従事．