Ambient Calculusを用いた物流検査システム

森本大輔[†] 加藤暢[††] 樋口昌宏[†]

物流の世界では，貨物の流通量が増加する一方，コンテナ管理の重要性が高まっている．我々はその解決策として Ambient Calculus による物流システムの形式的記述に基づく物流管理システムの構築について研究を進めている．物流システムは，荷物，コンテナ，コンテナ船というように，複数のより小さなパッケージを収容したパッケージがより大きなパッケージに収容されるという階層構造を持っている．一方 Ambient Calculus は，階層構造を持ち動的に構造が変化するシステムを形式的に記述するための言語であり，この特徴から物流システムの持つ階層構造を簡潔に表現することができる．さらに Ambient Logic の公理系を用いて記述の正当性を形式的に検証することが可能である．本論文では，実際のコンテナ輸送で使われる Excel 形式の送り状をもとに各コンテナの搬送経路を表現する Ambient Calculus 式を自動生成するシステム，Ambient Calculus 式をもとに実際の貨物の動きが式の遷移と適合しているかどうかを検査するシステムについて述べる．また，オンボード Linux を装備した箱をコンテナに見立ててこれらのシステムの有効性を調べる実験を行った．その結果についても報告する．

A Conformance Decision System for Physical Distribution with the Ambient Calculus

Daisuke Morimoto,[†] Toru Kato[††] and Masahiro Higuchi[†]

In the field of distribution, an increasing of the handling fault of luggage becomes serious problem while the amount of circulation of the freight increases. To solve the problem, we are researching on the constructing the distribution management system based on descriptions of the physical distribution system in the Ambient Calculus. The physical distribution system has the layered structure, e.g., packages (e.g., container) including smaller packages (e.g., luggage) are accommodated by larger entity such as container ships and so on. On the other hand, the Ambient Calculus is a formal description language suitable for representing systems whose layered structure dynamically changes. This feature enables us to express concisely physical distribution systems. The validity of the description can be formally verified with a system of axioms for Ambient Logic. We propose the automatic generating system of the Ambient Calculus formulae, and conformance decision system for the movement of actual freights. The former system generates the formulae that express the transportation route of containers based on the shipping invoice written in Excel files used in an actual container shipping. The latter system checks the movement of an actual freight by comparing it with transition of the formulae. We also present the result of simple experiments to show the effectiveness of those systems.

1. はじめに

近年，国際物流の取扱量は大型コンテナ船の就航などにより，ますます増大している[1),2)]．取扱量の増加とともに，貨物の取り違え防止や，（特に 9.11 テロ以降）貨物に対するセキュリティへの要求なども今まで以上に厳しくなっており，コンテナ管理の必要性が高まっていく中で，どのようにコンテナを管理していくかについて，物流業界では様々な方法を模索している．

現状のコンテナ管理は，位置情報や内容の情報などを取得し，トレーサビリティ機能やトラッキング機能などで行われている．たとえば，大手海運会社では RFID を用いた位置情報の取得などを実験している[3)]．また，2007 年 3 月から海上コンテナに IC タグを取り付けて，位置追跡を行う実験が始まる予定である[4)]．この実験は，米国土安全保障省，国土交通省の協力の下，横浜港で行われるもので，海上コンテナに GPS 機能を持った発信可能な IC タグを取り付け，海上や

† 近畿大学大学院総合理工学研究科
Interdisciplinary Graduate School of Science and Engineering, Kinki University
†† 近畿大学理工学部情報学科
School of Science and Engineering, Department of Informatics, Kinki University

陸上での位置を把握するものである．この実験の結果，効果が高いと判断されれば，これからのコンテナ管理の標準的な手法として採用される可能性が高い．しかし物流システムは，荷物，コンテナ，コンテナ船というように，複数のより小さなパッケージを収容したパッケージがより大きなパッケージに収容されるという階層構造を持っており，コンテナの位置情報のみに着目したこれらの検査方法では，物流システムの持つ階層構造がどのように変化しながら貨物が輸送されたのかという，より厳密な検査を行うことができない．そこで我々は Ambient Calculus [5] を用いた形式的記述に基づく物流システムのモデル化を行い，このモデルに沿って物流システムを管理する検査システムについて研究を進めている．Ambient Calculus を用いた物流システムのモデル化には，次のようなメリットがあると考えられる．

［モデルに沿った移動検査］ 物流システムは，2.2 節で示す送り状と呼ばれる書類によってその構造が規定され，送り状をもとに作られる指示書によって動作が規定される．したがって，送り状とは物流システムにおける仕様書に相当する．この送り状をもとに，階層構造を持った物流システム全体を Ambient Calculus のプロセス式としてモデル化し，実際の物の移動と式の遷移を対応させることで，このモデル（したがってもとの送り状）に沿ってコンテナなどの移動が行われているかどうかという検査が可能となる．その際，階層構造の変化にまで着目した検査を行うため，位置情報の検査だけではとらえきれなかった動作（不正な積み降ろしや積み替えなど）の検出も可能となる．前述の実験などで行われている IC タグを利用した位置情報取得によるコンテナ移動の検査と，モデル上での対応するプロセスの性質の比較とを組み合わせることによって，より高度なコンテナ管理が可能になる．

［トレーサビリティの向上］ コンテナなどがモデルに沿って移動を行っていれば，容易に貨物のトレーサビリティを実現することが可能となる．たとえば，運び終わった後でどのコンテナ船を利用したかが問題となったときに，元々のモデルを見ればどの道程をたどったかが判断できる．

［形式的検証］ 送り状をもとにプロセス式を生成し物流システムをモデル化することで，このプロセス式に対し Ambient Logic [6]（様相論理の一種）を用いたモデル検査を行うことが可能となる．これにより輸送が開始される前に，「いつかはこのコンテナがある場所にたどり着く」，「ある場所において決してコンテナが船から下ろされることはない」などといった荷主や運送業者が期待する性質をそのシステムが満たしているかといった検証が可能となる．また人為的な誤りにより送り状や指示書に内在する意図しないコンテナの移動を発見することが可能となる．

このように，階層構造の変化にまで着目した物流システムのモデル化を行うことで，コンテナ管理をより確実に行うことが可能だと考えられる．本論文では，実際のコンテナ輸送で使われる送り状をもとに，各コンテナの搬送経路を表現する式を生成し，コンテナやコンテナ船などの物の移動と，生成されたプロセス式の遷移を関連付けることにより，確実にコンテナが運ばれていることを検査するシステムについて述べる．また，システムを検証するため実験を行い得られた結果についても述べる．

2. 物流システム

この章では，貿易全般の流れと，そのモデル化について述べる．

2.1 貿易の要素

貿易には，取引の際に使われる “書類”，決済時に動く “代金”，実際に商品として取引される “物” の 3 つの要素がある [7]．

［書類］ 貿易は国を越えて行われる商取引である．売買契約を取り交わしても果たして商品が輸入者の所へ届くのか，どのようにその代金を輸出者は回収するのかなど，貿易には様々なリスクがつきまとう．少しでもそのリスクを軽減するために多くの書類が使われる．代表的な書類として，貨物の所有権を表す船荷証券，出荷案内としての役割を果たす送り状，通関作業関連の書類などがある．これらの書類は業務上必要不可欠なものであるが，実際の貿易実務ではこれらのやりとり，また転記作業などで大きなコストがかかっている．そこで，近年は電子データとして取り扱うことが進められている．代表的なものとして，Bolero.net [8] や NACCS [9] などがある．

［代金］ 貿易は物が届くまでかなりのタイムラグがあるため，どの時点で決済を行い代金を回収するかについて問題が生じる．現在，その解決に使われる手段は途中に銀行を介在させることで，代金回収のリスクを減らす信用状取引が主流である．これも電子化の流れが進んでおり，代表的なものとして SURF，TEDI [10] などがある．

［物］ 貨物の輸送方法は海上輸送と航空輸送に分かれ，海上輸送はその形態により 2 つに分類される．1 つは貨物をそのまま船の上に積み上げて輸送する在来船での輸送．もう 1 つはコンテナに詰め込み輸送す

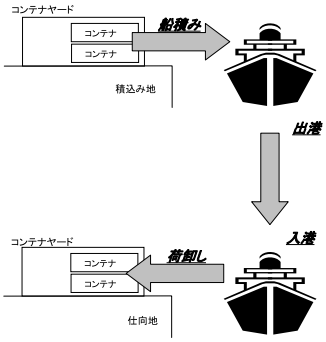

図 1 FCL での貨物輸送
Fig. 1 Transport on FCL.

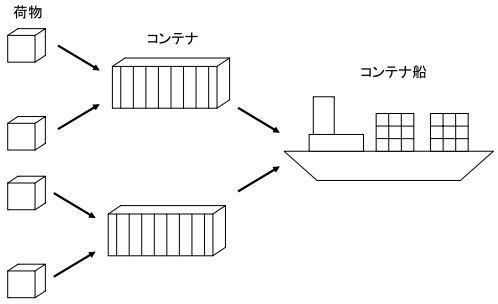

図 2 物流システムの持つ階層構造の例
Fig. 2 Example of physical distribution with hierarchical structure.

るコンテナ輸送である．コンテナ輸送はさらに，1 荷主の貨物で 1 つのコンテナをすべて使う FCL（Full Container Load）と，複数荷主の貨物を混載する LCL（Less than Container Load）に分かれる．

2.2 FCL での貨物輸送

実際の物の動きについて，FCL の場合を**図 1** に沿って述べる．輸出入の際には税関での作業が必要になる．この際に用いられる書類の中に送り状がある．これは，輸出者から輸入者に向けての出荷案内書である．ただし，輸出明細書，代金請求書，梱包明細書などの役割を果たすときもあり，関税法第 68 条により提出が義務づけられている重要な書類である．後に述べる検査システムで必要な式の自動生成には，この送り状を利用する．

一般的な FCL での貿易を行う場合，輸出を行おうとする者が自分でコンテナに貨物を積み込み，コンテナヤードまで運び込む．運び込まれたコンテナは，コンテナヤードで輸出にともなう税関手続きを行う．税関手続きが終わったコンテナは，載せるべき船が到着するまでしばらくコンテナヤードで待つことになる．搭載すべきコンテナ船が港に到着すると，コンテナはコンテナ船に積み込まれる．コンテナ船にすべてのコンテナが積み込まれると，港を出港し目的地へと向かう．目的地である港に到着したコンテナ船は，降ろすべきコンテナをコンテナヤードに積み出す．コンテナヤードに運び込まれたコンテナは，輸入にともなう税関手続きの後，輸入者に引き渡される☆．

これらの動作は様々な書類に従って行われているが，そのもととなっているのは送り状である．

2.3 物流システムとそのモデル化

物流システム（ここではコンテナ船を使った FCL 海上輸送システムを例にとる）は，動的に構造が変化する階層構造を持っている．**図 2** を用いて，物流システムの階層構造について述べる．コンテナヤード内にはすでに荷物が積み込まれたコンテナが荷主によって運び込まれている．そこにコンテナ船がやってくると，コンテナヤードから運ばれコンテナ船へと船積みされる．コンテナ船に積まれるコンテナの数は，コンテナ船によって様々であるが，数百〜数千個である（近年就航した超大型コンテナ船は 6,000〜8,000 TEU ☆☆の積載量を誇る）．その後コンテナ船は港を出港し，目的地となる港に向かう．港に到着したコンテナ船は，積んでいたコンテナをコンテナヤードに運び，受け取り人はそのコンテナを受け取る．物流システムは，このように小さなパッケージを含むパッケージが，さらに大きなパッケージに収容されるという階層構造と，その構造が変化していくという特徴を持っている．この特徴と，3 章で述べる Ambient Calculus の特徴は共通する点が多く，容易に物流システムをモデル化することが可能である．

現在の物流システムでは，コンテナを運ぶ運送業者，船会社などがばらばらにそれぞれの移動情報などを管理している．物流システム全体のモデル化を行い，すべての階層にわたり形式的にそれらを表現することで，階層構造の変化などにも対応した一元的な管理が可能となる．本論文では FCL 海上輸送システムについて，Ambient Calculus を用いて物流システムのモデル化を行い，モデルに基づいた貨物の移動と，実際の貨物の移動との整合性を検査するシステムについて提案しその実装について述べる．

3. Ambient Calculus

Ambient Calculus は，Microsoft Research の Luca

☆ コンテナヤードで貨物を引き渡す，輸入者が自分の工場までコンテナを運ぶなど，実際には様々な条件が売買契約の段階で決められている．

☆☆ TEU はコンテナ船の積載量を表す単位．20 フィートコンテナの容量が 1 TEU．

Cardelli と Andrew D.Gordon によって開発されたプロセス代数であり，動的な階層構造を持つシステムを形式的に記述するための言語である．この特徴から物流システムの持つ階層構造を簡潔に表現することができる．さらに，様相論理の一種である Ambient Logic の公理系を用いて，プロセス式そのものが意図した性質を持っているかどうかを検証することが可能である．本章では，ambient の直観的な説明を行い，Ambient Calculus の構文規則などの定義，プロセスや遷移の例などを示す．

3.1 Ambient

ambient とは「境界を持った場」という意味である．これには物理的な場と，論理的な場の両方が含まれる．ambient は次の性質を持っている．

- 各 ambient はそれぞれ名前を持ち，それによって識別される．
- ambient は入れ子構造にすることができる．ある ambient 内に別の ambient が存在している場合，前者を “親”，後者を “子” と呼ぶ，また，これ以降では親，子，祖先，子孫といった木構造を表す言葉を一般的に使う．
- ambient 内において，ローカルな処理を行うことができる．
- ambient は他の ambient に出入りすることが可能である．ある ambient が移動するときは，その子孫も階層構造を保ったままともに移動する．

3.2 構文規則・遷移規則

Ambient Calculus の構文規則と遷移規則は文献 5) で以下のように定義されている．

定義 3.1（構文規則） [5)]

$P, Q ::=$ *processes*

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

$M, N ::=$ *capabilities*

x	*variable*
n	*name*
in M	*can enter into* M
out M	*can exit out of* M
open M	*can open* M
ε	*null*
$M.N$	*path*

□

定義 3.2（遷移規則） [5)]

in

$n[in\ m.P \mid Q] \mid m[R] \rightarrow m[\ n[P \mid Q\ ] \mid R]$

out

$m[\ n[out\ m.P \mid Q] \mid R\ ] \rightarrow n[P \mid Q] \mid m[R]$

open

$open\ n.P \mid n[\ Q\ ] \rightarrow P \mid Q$

communication

$(x).P \mid \langle M \rangle \rightarrow P\{x \leftarrow M\}$

□

例 3.3（プロセス式と遷移例） コンテナ船に積まれているコンテナが船から出ていき，さらにコンテナヤードの中に入るという動作を例にとって，Ambient Calculus のプロセス式とその遷移について説明する．

$$ship[\ cont[\ out\ ship.in\ cy\ ]\ ]|cy[\ ] \quad (1)$$

$$\xrightarrow{out\ ship} ship[\ ]|cy[\ ]|cont[\ in\ cy\ ] \quad (2)$$

$$\xrightarrow{in\ cy} ship[\ ]|cy[\ cont[\ ]\ ] \quad (3)$$

式 (1)，(2)，(3) は定義 3.1 の構文規則に従って記述したプロセス式である．定義 3.1 の *inactivity* を表す “0” は省略可能である．*capability* を “.” でつなげて書くことが可能（*path*）であるが，これも本来は最後に “0” が続き，この例では省略している．

［構造について］ 式 (1)，(2)，(3) の中で，*ship* はコンテナ船を，*cont* はコンテナを，*cy* はコンテナヤードをそれぞれ表す ambient である．式 (1) での親子関係は，コンテナ船とコンテナヤードが並列に存在し，コンテナがコンテナ船の子になっている．このように *ship* や *cont* のような具体的な物を表す ambient のことを，これ以降 “物を表現する ambient” と呼ぶ．それ以外に，4.1 節で述べる物を表現していない制御用の ambient も存在するが，それらと，式の上での区別は特につけない．

［遷移について］ *out* は ambient を親 ambient の外へ移動させる能力，*in* は ambient を他の ambient の中へ移動させる能力を表現している．最初の状態である式 (1) では *ship* の中に *cont* があり，最初の遷移が行われた式 (2) では *ship* の中にあった *cont* が *ship* の外へと移動し，次の遷移が行われた式 (3) は *cont* が *cy* の中へと移動した．他の動作として，ambient の境界を消滅させる *open* がある． □

3.3 Ambient Calculus によるモデル化の利点

定義 3.1 で示した構文規則を見て分かるように，Ambient Calculus のプロセス式は ambient の階層構造

を持つ．これを利用することで例 3.3 で示したように，Ambient Calculus で書かれた式は物流システムの持つ階層構造を適切に，かつ直観的に表現することができる．また他のプロセス代数と違い，定義 3.2 で示した Ambient Calculus の遷移規則は物の出入りをプロセスの遷移と見なすものであり，例 3.3 で示したように物流システムにおける船や貨物の動きを直観的にとらえることが可能である．

このように，Ambient Calclus を用いることにより，物流システムの性質（階層的な構造や動作）を直観的に理解可能な形でモデル化できる．そのうえで，プロセス代数としての形式的な取扱い，たとえば後ほど述べる貨物の動きと式の遷移との関連付けによる貨物輸送の管理や，*composition* を用いた式の並列合成によるスケーラビリティなどが利点としてあげられる．この場合のスケーラビリティとは，物流に使われる船やコンテナ，港などがいくら増えようとも，処理系をそれらに載せるだけで，検査システム自体を変更しなくてもよいという意味である．

4. 検査システム

我々はすでに Ambient Calculus を動作させる処理系[11] を開発している．これは Ambient Calculus で書かれた式を実際に遷移させることにより，現実の物の動作をシミュレートする能力を持つ．この処理系をもとに，我々は荷物の運送状態が正しいかどうかを検査するシステムを開発した．検査システムの目的は以下のとおりである．

(1) 荷物を収容したコンテナや，コンテナを積んだコンテナ船が正しく移動することを確かめる．
(2) 正しくない移動を行った場合に知らせる．
(3) 現在のコンテナやコンテナ船の位置を把握する．

それぞれの目的を達成するために以下の実装を行った．

- 送り状からの式自動生成システム（5 章で述べる）
- 制御用 ambient を用いた，物を表す ambient の移動の制御
- 物を表す ambient ごとの分散処理
- 物の動きと式遷移との関連付け
- 上記関連付けにともなう動作エラーの警告
- 全体式の表示

4.1 制御用 ambient

物流システムをモデル化した Ambient Calculus のプロセス式は，コンテナ船やコンテナなど，実際の物を表す ambient と，その移動を制御するための制御用の ambient からなる．ここで制御用 ambient について説明する．例 3.3 では，すべての ambient は実際の物（コンテナやコンテナ船，コンテナヤード）を表現していた．しかし，それらと移動のための *capability* だけですべての遷移を正確に記述することは不可能である．移動を行うには順番が必要だからである．このことについて，式 (4) を例に説明する．

$$ship[in\ port.out\ port] |\ port[cont[in\ ship]] \quad (4)$$

式 (4) では，最初に *ship* が *port* に入る遷移が行われる．しかしこのままでは，次に可能な動作は *cont* の持つ *in* 動作（*cont* 内の *in ship* による遷移）と *ship* の *out* 動作（*ship* 内の *out port* による遷移）の 2 つであり，Ambient Calculus の持つ非決定性に基づく問題が生じる．後者が選ばれた場合，*ship* が先に *port* を出てしまい *cont* が *port* に取り残される．コンテナが積み込まれてから船が港を出るように式 (4) 内の各 ambient の動作を制御するためには，式 (5) に示すような制御用の ambient が必要となる．式 (5) の a，x，y，z が制御用 ambient である．

$$\begin{aligned} &ship[in\ port.x[out\ ship.in\ cont.open\ a] \\ &\qquad |open\ z|y[out\ port]] \\ &|\ port[cont[a[in\ ship.z[out\ cont.open\ y]]|open\ x]] \end{aligned} \quad (5)$$

これらの制御用 ambient により，*ship* が *port* に入った後，*cont* が *ship* に入ってからでないと *ship* は *port* から出られないようになる．例 5.1 では，具体的な例を用いて制御用 ambient の働きを詳細に説明する．

4.2 分散処理

検査システムには後述するプロセス式自動生成システムにより作られた式を解釈し，遷移させるプログラム（処理系）が必要である．処理系は，コンテナ，船，コンテナヤード，港のそれぞれに分散配置された計算機内に置かれ，親子関係（港の子としてコンテナヤードを設定するなど）を構築する．これにより，コンテナ名，船名，コンテナヤード名，港名などを名前とする ambient ができる．このときそれぞれの ambient の名前は，元々決まっている名前（港名や船名は当然決まっている．コンテナ名はコンテナ番号が ISO の規格☆により一意になるように定められている）であるため，一意なものになる．また，コンテナヤード名はすべて CY としているが，それらは特定の決まった港と親子関係を構築するので，混乱が生じることはない．また，最上位（港のさらに親の立場）から管理する ambient を設定する．港 ambient がどこに存在してい

☆ ISO6346-1995

るかについては，現実世界をどのようにモデル化するかの議論が必要になるが，本検査システムはコンテナ船による国際貿易を対象にしているため，海（SEA）という ambient を考え，その中に存在していると考えた．同じく，コンテナ船も最初の状態ではどこの港にも存在しておらず，海の上にいると考え，海 ambient の中にコンテナ船 ambient が存在している☆．

各コンテナについては，最初の状態ではコンテナヤードにあると考え，コンテナがコンテナヤードの子になるように設定している．各コンテナもそれぞれ処理系を持つ．6 章で述べる実験では，コンテナ用の処理系のために Linux ボードを用いた．各計算機間の通信には IP アドレスを用いているが，実際に運用する場合には，クローズドなネットワークを使うと考えられる．

処理系が複数存在するため，どの処理系がどの式を管理するのかを決めなければならないが，本システムでは，物を表現する ambient 内にある式については，その ambient の処理系が管理するようにした．たとえば，コンテナ船の処理系は，船内に積まれたコンテナの処理系と協調してコンテナ ambient の移動を管理し，コンテナ内の ambient はコンテナの処理系が管理する．したがってコンテナ船の処理系は，コンテナ内の ambient については関与しない（コンテナ船は，積まれているコンテナの名前は知っているが，その内部に何があるかについては知らないということになる）．このように，できるだけ式を物理的な階層構造に沿った形で分割することで，それぞれの処理系が管理する式を小さくすることができる．

ところで，実際の物（コンテナやコンテナ船など）が移動すると，その物を表現する ambient が現実の移動と同じように遷移可能であるかどうかの検査が，プロセス式をもとに行われるが，その際に通信が発生する（4.4 節参照）．たとえば，コンテナがコンテナ船に積まれる in 動作である場合，コンテナ船の処理系とコンテナの処理系の間で通信が発生する．物流システム全体を表現する式を 1 つのサーバで管理していた場合，様々な船の上での多くのコンテナ移動に関する通信が 1 つのサーバに集中する．分散処理を行うことにより，このような通信の集中を避けることができる．

また，1 つのサーバで集中管理していた場合，気象状態など何らかの理由でサーバとの通信ができなくなる可能性があり，コンテナ移動の検査が不可能になる恐れがある．各コンテナに処理系を実装することで，コンテナが船や港と通信できない状態でも自分の情報を自分で管理し続けることができ，コンテナの移動の正当性を各コンテナが自律的に判断することができる．

4.3 計算機間を移動する制御用 ambient の遷移

物を表現する ambient ごとに処理系が存在するため，制御用の ambient は計算機間を移動する必要がある☆☆．in 動作と out 動作について分けて考える．どちらの動作も，それらの遷移が可能かどうかが親の処理系（港の中でコンテナが船に入る場合，港の処理系が船とコンテナの親の処理系である）で検査され，遷移のための通信が行われる．

- in 動作

in 動作が発生し，かつ計算機を移動する遷移となるのは，in 動作を行う制御用 ambient が，物を表現する ambient と同じ親 ambient の中にあるときである．in 動作は以下の手順で行われる．

(1) 親の処理系は，in 動作を行う制御用 ambient と物を表現する ambient が両方とも自分の子であることを調べる．さらに，in 動作の移動先（in 先）の名前が正しいか，つまり物を表現する ambient の名前と一致するか調べる．

(2) 正しければ in 動作を遷移させる．すなわち，in 動作が消費され，制御用 ambient の式が in 先の処理系に送信され，物を表現する ambient の子となる．

- out 動作

out 動作が発生し，かつ計算機を移動する遷移となるのは out 動作を行う制御用 ambient が，物を表現する ambient の子となっているときである．out 動作は以下の手順で行われる．

(1) out 動作を行う制御用 ambient が移動元となる ambient の子であり，かつ移動元の ambient が物を表現している ambient かを，物を表現する ambient を管理する処理系が調べ，さらに out 動作の移動元（out 元）の名前が正しいか調べる．

(2) 正しければ out 動作を遷移させる．すなわち，out 動作が消費され，制御用 ambient の式が，out 元の ambient のさらに親（つまり制御用 ambient の祖父 ambient）の処理系に送信され，制御用 ambient は祖父 ambient の子となる．

4.4 物の動きと式遷移との関連付け

3 章で述べたとおり，実際の物（今回のシステムではコンテナやコンテナ船など）を表現する ambient と，

☆ 初期状態での物を表す ambient の位置は，起動時に設定ファイルで指定している．

☆☆ たとえば，船が港に入港した場合，そのことをコンテナヤードに伝える制御用の ambient の移動などである．

物を表現しない ambient（制御用に書かれた ambient など）を構文上は区別しない．しかし実際に遷移をさせていくうえでは，物を表現する ambient の動作と，制御用 ambient の動作は区別する必要がある．すなわち，制御用 ambient の遷移は，遷移可能なときであればいつでも遷移してよいが，物を表現する ambient の遷移は，実際に物が動いたタイミングでのみ遷移する．そうでなければ，式のうえでは船が港に来ているのに，実際はまだ海の上ということが起こりうる．そのため，式の遷移と物の動きについての関連付けが必要となる．本検査システムでは，物を表す ambient の遷移と現実の物の移動の関連付けを行う手段としてバーコードとバーコードリーダを利用した．物を表現する ambient には名前を表すバーコードが付けられており，必要に応じて計算機にはバーコードリーダが接続されている．以下では，物の動きと ambient の遷移の関連付けをどのように行っているかについて述べる．

- in 動作

物を表現する ambient が，他の物を表現する ambient に入ろうとするときの動作である．たとえば，船が港に入港する，コンテナをコンテナ船に積み込むなどである．このとき，in 動作を行おうとする ambient が，正しい ambient に入ることを調べるために，バーコード読み込みを利用した．

in 動作の例を，コンテナをコンテナ船の中に移動させる例を用いて説明する（**図 3**）．

(1) in 先となる ambient（この場合はコンテナ船）のバーコードリーダにて，バーコード（バーコードに記された名前）を読み込ませる状態にしておき，移動する ambient（この場合はコンテナ）のバーコードを読み込ませる．このバーコードを読み取らせることで，現実の移動（コンテナ船にコンテナを積み込む）と，式の遷移（コンテナ ambient がコンテナ船 ambient の中に入る）を関連付ける（図 3 の (1)）．

(2) 読み取られた名前を持つ ambient の in 動作が正しいか，つまり in 先であるコンテナ船 ambient と，移動するコンテナ ambient が同じ親（この場合は港 ambient）の中に存在するかという検査が，コンテナ船の処理系から港の処理系へ依頼される（図 3 の (2)）．

(3) 移動するコンテナ ambient の中に，すぐに動作可能かつ in 先の名前（コンテナ船 ambient）が正しい in 動作（この場合は “in コンテナ船”）が記述されているかの検査が，港の処理系からコンテナの処理系に依頼される（図 3 の (3)）．

(4) そのような in 動作がコンテナ ambient の中に存在していることが確認されれば，in 動作を遷移させる．すなわち，in 動作が消費され，移動するコンテナ ambient が in 先のコンテナ船 ambient の中に移動する．

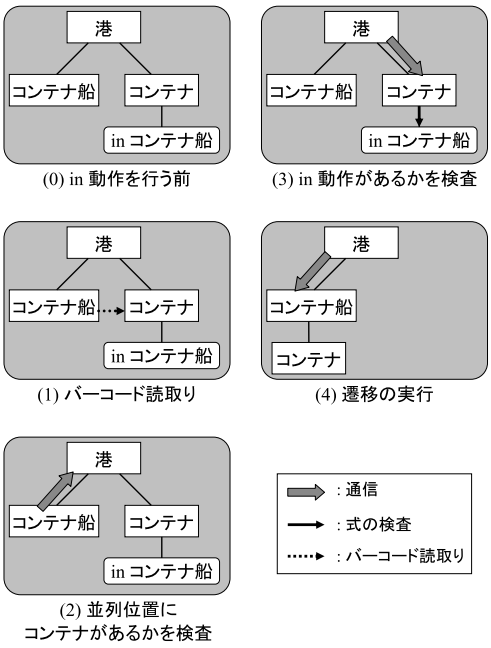

図 3 in 動作の例
Fig. 3 Example of in capability.

- out 動作

物を表現する ambient から，他の物を表現する ambient が出て行こうとするときの動作である．たとえば，港から船が出港する，コンテナ船からコンテナを降ろすなどである．このときも，out 動作を行う ambient が，正しい ambient から出てくることを調べるためにバーコード読み込みを利用した．

out 動作の例を，コンテナ船の中からコンテナを降ろす例を用いて説明する．

(1) out 元となる ambient（この場合はコンテナ船）のバーコードリーダで，バーコード（バーコードに記された名前）を読み込める状態にしておき，移動する ambient（この場合はコンテナ）のバーコードを読み込ませる．このバーコードを読み取らせることで，現実の移動（コンテナ船からコンテナを降ろす）と，式の遷移（コンテナ ambient がコンテナ船 ambient の中から出てくる）を関連付ける．

(2) 読み取られた名前を持つ ambient の out 動作が正しいかどうか（out 元であるコンテナ船 ambient の中に，移動するコンテナ ambient が存在するか）の検査が，コンテナ船の処理系で行われる．

(3) 次に移動するコンテナ ambient の中にすぐに動作可能かつ，out 元の名前（コンテナ船 ambient）が正しい out 動作（この場合は “out コンテナ船”）が記述されているかの検査が，コンテナの処理系で行われる．

(4) そのような out 動作がコンテナ ambient の中に存在していることが確認されれば，out 動作を遷移させる．すなわち，out 動作が消費され，移動するコンテナ ambient が out 元のコンテナ船 ambient の外に移動する．

以上の実装によって，式の遷移と実際の物の動きとの関連付けを行い，正しい in 動作と out 動作をとらえることが可能となった．どちらの動作も式の遷移よりも実際の物（例ではコンテナ）が先に動き，それらが動く先（入ろうとする，もしくは出ようとするコンテナ船）で移動をとらえ，後に式の遷移との整合性を調べることになっている．実際の現場で使われる際には，コンテナ船にコンテナが入ろうとした際に，その移動が正しいかどうか確認が行われる仕組みである．これは，式に書かれている遷移が正しいことを前提とした動作となっている．つまり，正しい式に従うことで物の移動が正しいかどうかを判断する仕組みである．こうすることで，後に述べる式自動生成システムで作られる式と組み合わせられ，正しい動作を確認することが可能となる．逆に，この動作と式の遷移が正しく合わない場合，その動作は間違った移動であると考えることができる．この間違った移動が行われた場合，その旨を表示することにより正しくない動作であることを知らせることが可能となる．

4.5 全体式の表示

分散処理を行った結果，物を表現している ambient は自分の子 ambient たちについて，その内容を知ることはできない（分かっているのはそれらの名前である）．しかし，検査システム全体として式を把握できなくては，今どこに荷物があるのかについて，正しい情報を取得することができなくなってしまう．したがって，物を表現する ambient の中も含めて，動作中の全体の式を表示する必要がある．

そこで本システムでは全体式を表示することが求められた場合，表示を行う計算機，およびその子となっているすべての ambient の情報を式として表示する機能を実装した．最上位である，“海 ambient” が動いている計算機から実行した場合，検査システム全体の式を表示することが可能となる．

5. 式自動生成システム

物流システムを表現するプロセス式を自動生成するには，送り状，船の航路，コンテナのリストが必要であるが，自動生成で一番重要なのは，送り状である．すべての書類のもととなる送り状を入力として，式を自動生成する．送り状は使っている輸出業者により仕様が異なるため，それぞれの種類にあわせて生成システムのほうで対応する必要がある．

送り状から式を自動的に生成すると同時に，現場で使われる書類に自動変換を行うことも必要である．たとえば，コンテナを積み込む際には船積依頼書（輸出者が作成）からドック・レシート（海貨業者が作成）が作られるが，このときに間違いが発生すると取引そのものが破綻してしまう可能性がある．ほかにも，貿易書類は転記をともなう部分が多く，しかもわずかな違いで大きな損害が発生する．式の正しさに加えて，書類も正確に式から作成されることで，貨物輸送の正確さが高まる．送り状の項目には輸出業者，運ぶ荷物の種類や数など，多くの情報が書かれているが，実際に式の生成で使われるのは次の項目☆である．

- Vessel（荷物を積む船の名前）
- From（荷物の積み込み地）
- To（荷物の仕向地）

実際の荷物はコンテナに詰めて運ばれるため，さらにコンテナ名が必要となるが，その名前は送り状には記載されない．そのため仮の名前を着けて式を生成する．

自動生成を完全に行うためには，送り状のほかにも次のようなデータが必要になる．通常，コンテナ船は港から出港して，次の港に行けば終わりというものではなく，いくつかの港に寄りその過程で荷物の積み込みと積み出しを行う．しかし，送り状だけではコンテナ船がある港から別の港に進むことは分かっても，他にどの港に寄るかは分からない．そこで，コンテナ船の航路をデータとして入力する必要がある．また，1通の送り状が必ずしも 1 つのコンテナだけを扱っているわけではなく，大口の荷物となれば複数のコンテナを必要とする．先に述べたとおり，送り状ではコンテナについて特に触れていないため，複数のコンテナを管理するためのコンテナリストと呼ばれる書類が必要になる．これは，荷物を詰めたコンテナの数量などを記した書類である．現時点では，これらのデータが得

☆ ただし，それぞれの項目名は送り状の仕様によって異なることがある．

			INVOICE					P/O NO ;		test01
								L/C NO ;		0
Shipped per:			ShipA	0				CONTRACT NO.:		test-cont-01
From ;			KOBE	,Japan				on/or about;		00-Jan-1900
Messers.;			0					TO;		SHANGHAI

図 4 送り状の例
Fig. 4 Example of invoice.

```
SEA[
    KOBE[
        CY[  Co1[a[out CY.in ShipA.z[out Co1.open ctrlKOBEtoSHANGHAI]]|open b|open c]
           |open x
        ]
    ]
|   SHANGHAI[CY[]]
|   ShipA[   in KOBE.x[out ShipA.in CY | b[in Co1.open a]]
           |ctrlKOBEtoSHANGHAI[out KOBE.in SHANGHAI
               .afterArrived[c[in Co1.out ShipA.in CY]]]
           |open afterArrived|open z]
    ]
]
```

図 5 遷移過程：初期状態
Fig. 5 Transition of the process: start.

られなかったため，船の航路とコンテナリストについては固定し，式の生成を行っている．

また，ひとたび送り状をもとにプロセス式を作ってしまえば，現在送り状から作られている書類を式から自動変換することも可能となる．貨物を動かすために必要な様々な書類を式から自動生成することで，モデルに沿った貨物の動きを指示することが可能となる☆．

例 5.1 生成システムの例として，1 つのコンテナを ShipA という船を使い，港 KOBE から港 SHANGHAI へと送る例を考える．最初に送り状を作成する（**図 4** 参照）．

次に，自動生成システムに読み込ませて，式を生成する．このとき，同時に必要な分だけバーコードも作られる．式 (6)，(7)，(8) は図 4 の送り状から作られたプロセス式である☆☆．

Co1.amb (6)
$a[out\ CY.in\ ShipA.$
$z[out\ Co1.open\ ctrlKOBEtoSHANGHAI]]$
$|open\ b\ |open\ c;$

cy_KOBE.amb (7)
$open\ x;$

ShipA.amb (8)
$in\ KOBE.x[out\ ShipA.in\ CY$
$|\ b[in\ Co1.open\ a]\ ]$
$|ctrlKOBEtoSHANGHAI[out\ KOBE.$
$in\ SHANGHAI.afterArrived[$
$c[in\ Co1.out\ ShipA.in\ CY]\ ]\ ]$
$|open\ afterArrived\ |\ open\ z;$

これらの式をそれぞれの処理系に読み込ませることで，システムの動作準備が完了する．

式生成の仕組みは次のようになっている．送り状より読み込んだ内容から，それぞれのコンテナが運ばれる行き先の港と，出発元の港，それから積み込まれる船名が得られる．同じ経路，同じ船に積まれる荷物ごとに数を数え，必要な送り状をすべて読み込んだ後，制御用の ambient を追加して，コンテナ，出発元の港にあるコンテナヤード，コンテナ船ごとに式を生成する．

制御用の ambient について，式 (6)，(7)，(8) を使い説明する．最初の状態として，各コンテナは出発元の港（KOBE）にあるコンテナヤードにあり，コンテナ船は海の上にあるものとする（**図 5** 参照）．一番最初に遷移が行われるのは，式 (8) の $in\ KOBE$ である．この遷移により，コンテナ船は港 KOBE に入る．次に，式 (8) 中の x ambient が船を出てコンテナヤードに入る．この ambient は式 (7) にある $open\ x$ により開けられ，その中にある b ambient がコンテナの中に入る（**図 6** 参照）．

コンテナの中で，式 (6) 中にある $open\ b$ により b ambient も開けられ，その中にある $open\ a$ により式 (6) 中にある $out\ cy.in\ ShipA$ が遷移可能となる．この *capability* が遷移することにより，コンテナはコンテナヤードを出て，コンテナ船の中に移動する（**図 7** 参照）．コンテナ船の中に移動したコンテナは，式 (6) 中の z ambient にある $out\ Co1$ が遷移可能であるため，この

☆ この機能は現時点では未実装である．
☆☆ それぞれの式の最後に ";" が付いているのは，処理系において構文解析を行う際，文末を判断するためである．

```
SEA[
    KOBE[
        CY[  Co1[a[out CY.in ShipA.z[out Co1.open ctrlKOBEtoSHANGHAI]]|open b|open c
                |b[open a]]]
        |ShipA[  ctrlKOBEtoSHANGHAI[out KOBE.in SHANGHAI
                    .afterArrived[c[in Co1.out ShipA.in CY]]]
                |open afterArrived|open z]
    ]
|   SHANGHAI[CY[]]
]
```

図 6 遷移過程 1
Fig. 6 Transition of the processes: 1.

```
SEA[
    KOBE[ CY[]
         |ShipA[ Co1[z[out Co1.open ctrlKOBEtoSHANGHAI]|open c]
                |ctrlKOBEtoSHANGHAI[out KOBE.in SHANGHAI
                    .afterArrived[c[in Co1.out ShipA.in CY]]]
                |open afterArrived|open z]
    ]
|   SHANGHAI[CY[]]
]
```

図 7 遷移過程 2
Fig. 7 Transition of the processes: 2.

```
SEA[
    KOBE[CY[]]
|   SHANGHAI[   ShipA[   Co1[open c]
                        |afterArrived[c[in Co1.out ShipA.in CY]]
                        |open afterArrived
                ]
                |CY[]
    ]
]
```

図 8 遷移過程 3
Fig. 8 Transition of the processes: 3.

```
SEA[
    KOBE[CY[]]
|   SHANGHAI[    CY[Co1[]]
                |ShipA[]
    ]
]
```

図 9 遷移過程：終了状態
Fig. 9 Transition of the processes: end.

ambient がコンテナからコンテナ船に移動する．移動した z ambient は式 (8) 中にある $open\ z$ により開けられ，その中にある $open\ ctrlKOBEtoSHANGHAI$ が遷移可能となる．この *open* 動作によりコンテナ船が港 KOBE から出て，港 SHANGHAI へと遷移可能となり，コンテナ船は移動を開始する（**図 8** ではすでに港 SHANGHAI に入った様子を示している）．

移動が終了すると，$open\ afterArrived$ が遷移可能であるため，中にある c ambient がコンテナの中に入る．コンテナ内で c ambient も開けられ，中にある $out\ ShipA.in\ CY$ が遷移可能になり，コンテナがコンテナ船からコンテナヤードへと移動する（**図 9** 参照）．以上で，コンテナが運ばれたことになる☆． □

このように，自動的に作られた式でコンテナが目的地に運ばれることが確認できた．非決定的な動作が発生しうるのは，同じ名前の ambient が存在するとき☆☆であるが，コンテナ・コンテナ船・港のすべての名前が異なるため，意図しない遷移が発生することはない．よって，自動生成システムで作られた式は必ず，目的地までコンテナが運ばれるということを表している．

物の移動と，このようなプロセス式の遷移とを厳密に関連付けることにより，物が移動するときにはプロセス式の遷移どおりに動いていることになる．したがって本検査システムは，必ずコンテナが目的の港まで運ばれ，かつ，途中で意図しない動きをすることが

☆ 説明では省略したが，コンテナやコンテナ船が動く際には，4.4 節で述べた式の遷移と物の動きの関連付けが行われている．

☆☆ 同名の ambient があると，in 動作の目的地がどの ambient になるかが一意に決定できない．また制御用 ambient は同名であるものがあるが，これらは同名であっても意図しない遷移を引き起こすことはない．

```
Co1
    a[out CY.in ShipA.z[out Co1.open ctrlSHANGHAItoKOBE] ] | open b | open c;

Co2
    a[out CY.in ShipA.z[out Co2.open ctrlSHANGHAItoKOBE] ] | open b | open c;

Co3
    a[out CY.in ShipB.z[out Co3.open ctrlSHANGHAItoOSAKA] ] | open b | open c;

cy_SHANGHAI
    open x|open x;

ShipA
    in SHANGHAI.x[out ShipA.in CY | b[in Co1.open a] | b[in Co2.open a] ]
    |ctrlSHANGHAItoKOBE[ctrlSHANGHAItoKOBE[out SHANGHAI.in KOBE.afterArrived[
        c[in Co1.out ShipA.in CY]|c[in Co2.out ShipA.in CY]]]]
    |open afterArrived|open z|open z;

ShipB
    in SHANGHAI.x[out ShipB.in CY | b[in Co3.open a] ]
    |ctrlSHANGHAItoOSAKA[out SHANGHAI.in OSAKA.afterArrived[
        c[in Co3.out ShipB.in CY]]]
    |open afterArrived|open z;
```

図 10 生成された式（複数のコンテナを別々の港に運ぶ）
Fig. 10 Generated formulae (plural containers carry to each ports).

```
SEA[
    SHANGHAI[
            CY[ open x|open x
                |Co1[a[out CY.in SHipA.z[out Co1.open ctrlSHANGHAItoKOBE] ] | open b | open c]
                |Co2[a[out CY.in SHipA.z[out Co2.open ctrlSHANGHAItoKOBE] ] | open b | open c]
                |Co3[a[out CY.in SHipB.z[out Co3.open ctrlSHANGHAItoOSAKA] ] | open b | open c]
            ]
    ]
|   KOBE[CY[]]
|   OSAKA[CY[]]
|   ShipA[   in SHANGHAI.x[out ShipA.in CY | b[in Co1.open a] | b[in Co2.open a] ]
            |ctrlSHANGHAItoKOBE[ctrlSHANGHAItoKOBE[out SHANGHAI.in KOBE.afterArrived[
                c[in Co1.out ShipA.in CY] | c[in Co2.out ShipA.in CY]]]]
            |open afterArrived|open z|open z
    ]
|   ShipB[   in SHANGHAI.x[out ShipB.in CY | b[in Co3.open a] ]
            |ctrlSHANGHAItoOSAKA[out SHANGHAI.in OSAKA.afterArrived[
                c[in Co3.out ShipB.in CY]]]
            |open afterArrived|open z
    ]
]
```

図 11 システム全体の式
Fig. 11 Formulae for the whole system.

ないことを保証するシステムとなっている．

6. 実　　験

提案手法に基づいて本システムを実装し，その実効性を確認するために以下の条件で実験を行った．2 隻のコンテナ船 ShipA と ShipB があり，ShipA は港 SHANGHAI から港 KOBE へと向かい，ShipB は港 SHANGHAI から港 OSAKA へと向かうものとする．さらに，ShipA を使い 2 つのコンテナ Co1 と Co2 を，ShipB を使いコンテナ Co3 を運ぶという条件を想定した．

本実験では，3 つのコンテナを扱うため 3 枚の送り状が必要となる．各送り状から**図 10** のような式が作られる．Co1 と Co2 は港 KOBE に，Co3 は港 OSAKA に運ばれる．cy_SHANGHAI は港 SHANGHAI にあるコンテナヤードである．各プロセス式はそれぞれの処理系に読み込まれる．たとえば，cy_SHANGHAI は港 SHANGHAI のコンテナヤードにある処理系で読み込む，などである．また，遷移のための式を必要としない処理系には inactivity “0” のみからなる式を読み込ませる．

起動した状態での物流システム全体を表す式は**図 11** のようになる．ここでの，SEA，$SHANGHAI$，$KOBE$，$OSAKA$，$ShipA$，$ShipB$，$Co1$，$Co2$，$Co3$，そしてそれぞれの港における CY は，各処理系を起動したことにより現れた ambient であり，それらの処理系には前述した式を読み込ませている．

起動が完了した状態では，港 SHANGHAI のコンテナヤードに 3 つのコンテナがあることが確認できる（**図 12** 参照）．ここで，港 SHANGHAI のコンテナヤードにあるコンテナの中身が見えないのは，4.2 節で述べたように，子やそれ以下の階層にある物を表現する ambient については，中身を知らないからである．また図 12 は，最後の 2 行が “操作を入力してく

```
CYの親はSHANGHAI
Co3の情報を返します
子供のIPは192.168.0.3
Co1の情報を返します
子供のIPは192.168.0.20
Co2の情報を返します
子供のIPは192.168.0.1
登録しました
IPを登録しました
操作を入力してください
fire,step,exit
state
CY[open x|open x|Co3[]|Co1[]|Co2[]]
操作を入力してください
fire,step,exit
```

図 12 港 SHANGHAI のコンテナヤード初期状態
Fig. 12 Initial state of the containeryard on the port SHANGHAI.

```
move
バーコード入力
Co1

************************************************
                MOVE !!!
************************************************

Thread-4: connecting to object server8885
Thread-4: HORB object server8885 has been connected
Thread-4: Waiting for method call
Thread-4: objectID= server8885 classNo= 1 methodNo= 1002
[  Parser  ] at ambient
[  Parser  ] at process
Co1
Thread-4: objectID= server8885 classNo= 1 methodNo= 1013
Thread-4: objectID= server8885 classNo= 1 methodNo= 1003
操作を入力してください
fire,step,exit
state
CY[Co2[]|Co1[]]
操作を入力してください
fire,step,exit
```

図 13 移動完了（港 KOBE のコンテナヤード）
Fig. 13 End of the moving on containeryard in Port KOBE.

ださい (改行) fire,step,exit” と表示されている．この状態は，システムの待機状態である．システムの待機状態では，物の動きをバーコードで入力する状態に移行したり，現在の式の状態を表示するためのコマンドを入力したりできる．何かの動作を行うとこの状態に戻り，次のコマンドが入力されるまで待機状態となる．

式の最初の遷移は ShipA と ShipB が港 SHANGHAI に入るところから始まるが，どちらから動いてもかまわない．以下，コンテナ船やコンテナの移動と，それにともなう遷移についてのみ述べる．

コンテナ船が港 SHANGHAI に入港し，入港を伝える ambient が各コンテナに移動，
→ コンテナがコンテナヤードからそれぞれのコンテナ船に移動，
→ 移動を伝える ambient がコンテナ船内で遷移，
→ コンテナ船が港 SHANGHAI から出港，
→ShipA は港 KOBE へ，ShipB は港 OSAKA へ入港，
→ それぞれ，入港したことを知らせる ambient が各コンテナに移動，
→ambient の遷移によりコンテナはコンテナ船から出て，コンテナヤードに入る．

最終的にすべてのコンテナを運んだ状態が **図 13**（港 KOBE のコンテナヤード）と **図 14**（港 OSAKA のコンテナヤード）である．これらの図において，Co1 と Co2 は港 KOBE のコンテナヤードに，Co3 は港 OSAKA のコンテナヤードに移動したことが表示されている．

また図 13 と図 14 は物の動きと関連付けられた遷

```
move
バーコード入力
Co3

************************************************
                MOVE !!!
************************************************

Thread-3: connecting to object server8885
Thread-3: HORB object server8885 has been connected
Thread-3: Waiting for method call
Thread-3: objectID= server8885 classNo= 1 methodNo= 1002
[  Parser  ] at ambient
[  Parser  ] at process
Co3
Thread-3: objectID= server8885 classNo= 1 methodNo= 1013
Thread-3: objectID= server8885 classNo= 1 methodNo= 1003
操作を入力してください
fire,step,exit
state
CY[Co3[]]
操作を入力してください
fire,step,exit
```

図 14 移動完了（港 OSAKA のコンテナヤード）
Fig. 14 End of moving on containeryard in Port OSAKA.

移を行った図でもある．図 13 で，“move” と表示されている．これはバーコードによる読み取りを行うコマンドであり，キーボードより入力して “バーコード入力” の表示状態に移行することで，バーコードによる読み取りが行える状態になっている．その次に入力されている Co1 は港 KOBE のコンテナヤードに移動してきたコンテナの名前であり，バーコードより入力された名前である．次の表示（MOVE !!!）で，物の動きと関連付けられた遷移であることを表し，しばらく通信ログが流れた後，“操作を入力してください” と表示される状態に移行することで，移動が正しいことを確認し，遷移が行われたことを示す．正しくない

```
操作を入力してください
fire,step,exit
move
バーコード入力
Co3

************************************************
                MOVE !!!
************************************************

**** 式に無い移動です!! もう一度移動を確認してください!! ****
操作を入力してください
fire,step,exit
```

図 15 エラーメッセージ表示
Fig. 15 Error message.

（式にない）物の動きが検出された場合の表示を**図 15** に示す．このようにエラーメッセージが表示されることで，物の動きが式に記述されたものではないことが確認できる．

7. 結　　論

本論文では，Ambient Calculus 式によって記述されたコンテナなどの移動が，物の動きと式の遷移との関連付けにより，正しく運ばれていることを検査するシステムについて述べた．また，実際にプロセス式自動生成システム，プロセス式を解釈し遷移させる処理系を作成し，2 隻のコンテナ船により 3 個のコンテナを運ぶ条件下での実験を行うことによりシステムの有効性を示した．

今後の課題として，次の点があげられる．

［式作成時の情報不足］たとえば，船がどの港に寄港して進んでいくかは，送り状だけでは判断できない．また，コンテナについての情報は梱包明細書（Packing List）や，船積依頼書（Shipping Instruction）などから取得する必要がある．それぞれの書類も，輸出者が作成するものや，船会社，海貨業者が作成するものもあるため，多くの業者から情報を得る必要がある．

［式から書類への自動変換］5 章で述べたように，現場で必要なシステム動作書類をプロセス式から自動生成できる機能が必要である．

［Ambient Logic への対応］式生成システムにより作られたプロセス式に対し Ambient Logic の公理系を用いたモデル検査が行えるような検証系を構築する．これにより輸送が開始される前に，「いつかはこのコンテナがある場所にたどり着く」，「ある場所において決してコンテナが船から下ろされることはない」などといった物流システムそのものの性質の検証が可能となる．

謝辞 本研究に対し多くのご助言をいただきました近畿大学森山真光講師に深く感謝いたします．

参 考 文 献

1) 国土交通省：平成 17 年度国土交通白書．
2) 国土交通省海事局：平成 18 年度版海事レポート．
3) 月刊「マテリアルフロー」編集部：海上コンテナへの RFID 導入で貨物情報のリアルタイム管理へ，物流 IT・ID ハンドブック 2004/2005，流通研究社，pp.63–74 (2004)．
4) 海上コンテナの位置追跡．日本経済新聞 2007 年 1 月 24 日発行 夕刊 3 面．
5) Cardelli, L. and Gordon, A.D.: Mobile Ambients, *Theoretical Computer Science*, Vol.240, pp.177–213 (2000).
6) Cardelli, L. and Gordon, A.D.: Ambient Logic, *Mathematical Structures in Computer Science* (2003).
7) 片山立志：よくわかる貿易書類入門，日本能率協会マネジメントセンター (2006)．
8) Bolero International Limited: Bolero.net. http://www.bolero.net/
9) 独立行政法人通関情報処理センター：NACCS. http://www.naccs.go.jp/
10) TEDI Club: TEDI. http://www.tediclub.com/
11) Kato, T., et al.: The Implementation of Ambient Calculus with HORB for Mobile Agents, *Proc. 7th World Multiconference on Systemics, Cybernetics and Informatics*, Vol.2 (2003).

(平成 18 年 12 月 15 日受付)
(平成 19 年 3 月 11 日採録)

森本　大輔

2007 年近畿大学大学院総合理工学研究科エレクトロニクス系工学専攻修了．同年 4 月新日鉄ソリューションズ（株）入社．プロセス代数を用いたシステムモデル化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．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．並行論理型言語の意味論，プロセス代数に対する等価性理論等の研究に従事．

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 年同大学院博士前期課程修了．（株）富士通研究所，大阪大学講師等を経て，2000 年より近畿大学理工学部助教授．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．