情報処理学会論文誌　数理モデル化と応用　Vol.14　No.1　21–32 (Jan. 2021)

多重Ambient Calculusのための統合開発環境

加藤 暢[1,a)]　山田 瞭太[1]　鳥山 颯斗[1]　大倉 亮介[1]　樋口 昌宏[1]

受付日 2020年8月28日, 採録日 2020年9月29日

概要：本稿では，多重 Ambient Calculus（MAC）のための統合開発環境を提案する．MAC とは，物流システムのモデル化に特化したプロセス代数である．コンテナを用いる海上物流では，多数のコンテナが数隻のコンテナ船に載せ替えられながら，何箇所かのハブ港を経由し目的地に輸送される．我々は現在，コンテナ輸送が計画どおりに行われているかどうかを確認する物流監視システムについて研究を進めている．このシステムは，MAC を用いてモデル化された物流計画と，そのモデルの一部を書き込んだ RFID タグを用いて検知した実際のコンテナの移動を比較することで，コンテナが物流計画どおりに取り扱われているかどうかを確認するものである．このような監視活動を行うためには，物流システムの持つ動的な階層構造を表現する MAC のプロセス式を正確に記述する必要がある．本稿で提案する統合開発環境 IDE4MAC は，通常の編集機能に加え，複数の動作が選択可能なプロセスに対する選択実行機能や巻き戻し機能など，MAC のプロセス式の持つすべての非決定的な動作を確認するために有用な機能を備える．さらに，プロセス式の階層構造を表す木構造の図を編集する機能，および木構造の図とプロセス式のリアルタイムな相互変換機能を備える．IDE4MAC を使用することで，大規模かつ複雑な経路を持つ輸送システムのモデル化が容易になり，物流監視システム開発の効率を上げることが可能となる．IDE4MAC は，インストールや使用時の利便性を考慮し，Eclipse のプラグインとして開発している．

キーワード：統合開発環境，プロセス代数，Ambient Calculus，海上物流システム

The Integrated Development Environment for the Multiple Ambient Calculus

Toru Kato[1,a)]　Ryota Yamada[1]　Hayato Toriyama[1]　Ryousuke Okura[1]　Masahiro Higuchi[1]

Received: August 28, 2020, Accepted: September 29, 2020

Abstract: We propose an integrated development environment (IDE) for developing formulae written in the multiple ambient calculus (MAC). MAC is a kind of process algebra designed for modeling freight systems in which a lot of containers are transported by several vessels from one port to another port via several hub ports. We have been studying a handling management system that confirms the validity of container handling during shipping. The system checks whether containers are being correctly handled by comparing the movement of the containers, which is sensed by radio frequency identification (RFID) tags, with formal models (formulae) written in MAC. For accurate and automatic management, we need to develop MAC formulae that accurately express the nested and dynamic changing structure of the entire freight system (ports, vessels, and containers) and the movement of those objects. The IDE presented in this paper (IDE4MAC) equips ordinal editing functions and several debugging functions such as a selective execution function and backward tracing function that are useful for checking all non–deterministic actions of MAC processes. IDE4MAC also has a graphical editor by which we can edit tree structures expressing nested structure of MAC processes and the real-time mutual conversion function between process formulae and tree structures. IDE4MAC enables us to easily model freight systems with large scale and complicated paths, that improves the efficiency of developing the handling management system. We developed IDE4MAC as plugins of Eclipse so that it can be easily installed and used.

Keywords: IDE, process algebra, ambient calculus, freight system

1. はじめに

本稿で提案する統合開発環境（IDE4MAC）は，現在研究を進めている物流監視システム [13] で使用する多重 Ambient Calculus（MAC）[5] のプロセス式設計支援を主な目的として開発している．この物流監視システムは，物流システム全体を MAC のプロセス式でモデル化し，RFID タグに書き込んだプロセス式の遷移と，RFID 機器で読み取った実際のコンテナの動きを比較することで，コンテナが輸送計画どおりに扱われているかどうかを自動的に確認するものである．

コンテナ海上輸送において，大規模なコンテナターミナルでは年間に取り扱うコンテナ数は 100 万を超えるが，現在多くのターミナルでは，コンテナ船への積込みや積降ろしの最終確認は係員が目視で行っている．これに対し，RFID などを用いた最終確認の自動化に関する研究が多くの機関で行われているれている [15], [16], [17], [20], [21]．このように，コンテナの輸送を RFID タグにより管理する仕組みは近年高い関心を集めており，一部は実際に運用が開始されている．

これらに共通する考え方は，タグの ID とコンテナ情報をデータベースなどで紐付けしておき，リーダで読み取った ID をネットワークを介してデータベースを持つサーバに送り，個々のコンテナの動きをサーバ側で監視するというものである．したがってこれらは，無線 LAN やインターネットなどネットワーク環境がつねに活用できることを前提とした仕組みになっている．

これに対し我々が研究している物流監視システムでは，コンテナ情報の管理を中央のサーバに頼るのではなく，タグそのものに書き込んだ情報を元にコンテナの取扱いを監視する方法を採用している．これにより，数千個のコンテナの取扱ごとに発生する数千回の中央サーバとの通信を削減することが可能となる．

MAC が基礎としている Ambient Calculus（AC）[1] は，動的な階層構造を代数式で表現することができ，ネットワーク間を移動するようなモバイルプロセスの記述に特化したプロセス代数である．一方海上物流にも，大きな枠組みであるコンテナ船が小さな枠組みであるコンテナを積んでいるという階層構造が存在している．そしてその階層構造は，船がある港から別の港へと移動したり，コンテナの積込みや積降ろしをするとき動的に変化する．

我々は AC と物流システムの双方が持つ動的な階層構造という類似点に着目し，文献 [9] において，物流書類の内容を AC のプロセス式を用いて表現し，実際の物流がそのプロセス式の記述どおりに行われているかを監視するシステムを提案した．さらに，現実の物流で頻繁に生じるコンテナの追加，キャンセルなど動的な変更にも柔軟に対応できるモデル化を行うため，我々は文献 [5] において MAC を提案し，文献 [3] や文献 [13] [*1] において MAC を用いた物流監視システムを提案した．これらのシステムは，エクセル形式で記述された物流書類から MAC のプロセス式を生成する機能，MAC の式を遷移させることのできる処理系，および RFID 機器を用いて検知した実際のコンテナの取扱いをプロセス式の遷移と対比させ，その取扱いが正しいものであるかどうかを監視する機能からなる．

このような監視活動を行うためには，物流システムの持つ動的な階層構造を表現する MAC のプロセス式を正確に記述する必要がある．IDE4MAC は，通常の編集機能に加え，複数の動作が選択可能なプロセスに対する選択実行機能や巻き戻し機能など，MAC のプロセス式の持つ非決定的な動作を確認するために有用な機能を備える．さらに，プロセス式の階層構造を表した木構造の図を編集する機能，および木構造の図とプロセス式のリアルタイムな相互変換機能を備える．IDE4MAC を使用することで，大規模かつ複雑な経路を持つ輸送システムのモデル化が容易になり，物流監視システム開発の効率を上げることが可能となる．

本稿ではまず 2 章で MAC による物流システムのモデル化の例を示す．3 章では本研究で提案する IIDE4MAC の持つ各種機能について説明する．4 章では IDE4MAC を用いた物流システムを表すプロセス式の設計とデバッグの実例について述べ，5 章では IDE4MAC が大規模なプロセス式生成にも耐えられることを説明する．6 章ではプロセス代数の統合開発環境として有名な PAT，他言語に対する既存のグラフィカル表現機能を備えた開発環境，デバッガとの比較を行う．

2. 多重 Ambient Calculus（MAC）

本研究では構文規則と遷移規則に関し，AC については文献 [1] の定義を，MAC については文献 [5] の定義をそのまま利用している．本章では物流システムの簡単なモデルを用いて，これらの規則を直観的に説明する．

2.1 AC による物流システムのモデル例

ambient とは，人，建物，あるいは計算機内で実行されるプロセスなど，名前のついた対象物の構造や性質を名前付きの [括弧] で表現する AC 特有の構文要素である．入れ子状の [括弧] を用いて様々な対象物の階層構造が表現できる．式 (1) は，1 つのコンテナを東京港で船に積込み香港港へ移送するという物流システムを表したプロセス式であ

[1] 近畿大学理工学部情報学科
Kindai University, Kowakae, Higashi-Osaka, Osaka 577–8502, Japan
[a)] kato@info.kindai.ac.jp

[*1] 文献 [13] ではモデル検査システムも提案しているが，このモデル検査システムと IDE4MAC のデバッグ機能との関係については 4 章で述べる．

る[*2].

$$TK[co1[\ in\ SHIP.\ lcomp1[out\ co1]\ ]]\ |\ HKG[\ ]$$
$$|\ SHIP[in\ TK.open\ lcomp1.\ out\ TK.\ in\ HKG] \quad (1)$$

この式は，5 つの ambient：*TK*，*co*1，*lcomp*1，*HKG*，*SHIP* からなっており，*lcomp*1 以外の 4 つはそれぞれ東京港，コンテナ，香港港，船という現実のモノを表している．そして，*TK*，*HKG*，*SHIP* が並列合成演算子 | で結ばれていることから，各港と船が隣り合った位置に存在し，また，*co*1 ambient が *TK* ambient の [括弧] の内側に記載されていることから，コンテナが東京港の中に存在しているという物流システム全体の構造を表している．

ambient 内に現れる *in*，*out*，*open* は capability と呼ばれ，その ambient の動作を表現する．たとえば *in TK* capabilitiy を用いて式 (1) は式 (2) に遷移する．この遷移は船が東京港へ入港する動作を表している．

$$TK[co1[\ in\ SHIP.\ lcomp1[out\ co1]\ ]\ |\ SHIP[open\ lcomp1.\ out\ TK.\ in\ HKG]]\ |\ HKG[\ ] \quad (2)$$

式 (2) の *SHIP* ambient には，東京港からの出航を表現するための *out TK* capability が存在するが，その前に *open lcomp*1 が存在することにより *out TK* は実行できない．これは，コンテナの積込みが完了するまでは船が出航できない，という物流システムの制約を表している．*open* capability は，並行な位置にある ambient を消滅させるために用いられる．*co*1 が *SHIP* の中に入り，*co*1 内の *lcomp*1 ambient が *out co*1 を実行して *open lcomp*1 と並行な位置に来たときにはじめて *open lcomp*1 が実行され，*SHIP* ambient は *out TK* を実行できるようになる．

このように，AC ではプロセス式の構造で物流システムの構造を表し，capablity の実行や制約によって，物流システム内の対象物の動作や対象物間の同期的制約を表すことができる．

これらの式に現れる *lcomp*1 は，実際のモノを表す ambient ではなく，モノとモノの間に課せられる同期的制約を表すために使用される ambient である．このような ambient を**制御用 ambient** と呼ぶ．式 (1) の *lcomp*1 ambient は *in SHIP* に先行されているため，*out co*1 は実行されない．このように何らかの capability に先行されている ambient，およびその内部にある ambient は遷移に関与しない．そのような ambient を**非活性 ambient** といい，そうでないものを**活性 ambient** と呼ぶ．

[*2] 実際に使用している式では *TK*[*CY*[*co*1[...]...] のようにコンテナはコンテナヤード内に存在するという物流システムの構成を忠実に表現しているが，ここでは説明を簡単化するため省略している．

2.2　MAC による物流システムのモデル例

AC による物流システムのモデル化では，コンテナ数が増えると同期制御のための記述が複雑になり，コンテナの追加やキャンセルといった現実の物流システムで頻繁に起こる変化に柔軟に対応することが困難となる．MAC ではこの問題を，複数のプロセス式の組で物流システムをモデル化することで回避している [4], [5]．MAC の構文規則である全体式と個別式は文献 [1] で以下のように定義されている．

定義 2.1（全体式と個別式（MAC の構文規則））
$P_1, P_2, \ldots, P_n$ をそれぞれ AC のプロセスとする．このとき $\overline{P} = (P_1, P_2, \ldots, P_n)$ を全体式，各 P_i（$1 \leq i \leq n$）を $\overline{P}$ の個別式，あるいは第 i 成分と呼ぶ．　□

簡略化した例を用いて定義 2.1 を説明する．式 (3) は，2 つのコンテナ *co*1 と *co*2 を東京港 *TK* で船 *SHIP* に積み込むという物流システムを表した全体式である．

$$\begin{pmatrix} TK[SHIP[co1[\ldots]\ |\ out\ TK.\ in\ HKG]]\ |\ HKG[\ ]\ , \\ TK[co2[\ in\ SHIP.\ lcomp[out\ co2]] \\ \quad |\ SHIP[open\ lcomp.\ out\ TK.\ in\ HKG]]\ |\ HKG[\ ]\ , \\ TK[SHIP[\ out\ TK.\ in\ HKG]]\ |\ HKG[\ ] \end{pmatrix} \quad (3)$$

式 (3) の 2 行目にある第 1 成分は，すでに *SHIP* に積み込まれているコンテナ *co*1 の動作を表わす個別式であり，3，4 行目にある第 2 成分は，積込み前のコンテナ *co*2 の動作を表す個別式である．また 5 行目にある第 3 成分が東京港から香港港へ向かう船の航路を表す個別式である．このように MAC では，各コンテナおよび船ごとに個別式という独立した AC のプロセス式でそれぞれの性質を記述する．これにより，コンテナの追加やキャンセルを個別式の挿入や削除で表現することが可能となる．

MAC では，ただ 1 つの個別式にのみ現れる ambient および制御用 ambient を**個別 ambient** と呼び，複数の個別式に共通して現れる，制御用 ambient 以外の ambient を**大域 ambient** と呼ぶ．式 (3) において，*co*1，*co*2 ambient はそれぞれ第 1 および 2 成分である個別式にのみ現れる個別 ambient である．また *TK*，*HKG*，*SHIP* は複数の個別式に共通に現れる大域 ambient である．以降，大域 ambient 名は先頭に大文字を用いて区別する．

MAC の遷移規則については，文献 [5] の定義をそのまま利用する．ここでは式 (3) に示した全体式を用いて遷移規則を直観的に説明する．式 (3) の第 1 成分では *SHIP* の *out TK* という遷移が可能な状態であるが，第 2 成分では *SHIP* の *out TK* の前にコンテナ積込み確認用の capability があるため *out TK* はブロックされている．第 2 成分で

*co*2 ambient が *SHIP* に入り *SHIP* の *out TK* が可能になったとき，すべての成分で *out TK* が可能になり，この時点で初めて *SHIP* の *out TK* という遷移が可能になる．このように MAC では，遷移規則によってすべてのコンテナが積まれた後に船が出航するといった同期的な動作を表現するため，AC のような複雑な記述は不要になる．

3. MAC のための統合開発環境

2.2 節で示したような MAC のプロセス式は，物流システムの構成やそのなかのモノの動きを直接的に表したものになっている．しかし，多くのコンテナが複数のハブ港を経由して複数のコンテナ船に載せ替えられながら輸送される様子をモデル化する式を emacs などの一般的なエディタで正確に記述することは困難である．そのような作業を支援する目的で我々は IDE4MAC を構築した．IDE4MAC は主に 3 つの部分，プロセス式編集用テキストエディタ（以下 TE），プロセス式の木構造を編集するグラフィカルエディタ（以下 GE），そしてプロセスを動作させるための MAC 処理系から構成されている．本章ではこれらの機能について説明する．IDE4MAC は，インストールや使用時の利便性を考慮し，エクリプスのプラグインとして Java 言語で開発しているため，Windows，Mac OS そして Linux 上で動作する[*3]．

実装規模はおおよそ以下のとおりである．TE：28 クラス（約 4,000 行），GE 63 クラス（約 6,300 行），MAC 処理系 65 クラス（約 15,000 行）．

3.1 テキストエディタ

図 1 は IDE4MAC のパースペクティブを示している．

通常の Java 用のパースペクティブと同様，IDE4MAC はプロセス式編集用の TE を中央に持ち（②），実行後のプロセス式（実行結果）を表示するコンソールがその下にある（③）．実行も Java の IDE と同様メニューバーにある MAC メニュー（①）から行う．

選択実行機能

プロセス代数の特徴として，AC や MAC では非決定的な動作が簡単に記述できる．たとえば，*TK*[ ]|*SHIP1* [*in TK*]|*SHIP2* [*in TK*] というプロセス式の場合，*SHIP1* が *TK* に入った後 *SHIP2* が *TK* に入る場合とその逆の 2 通りの実行順序がある．この順序は処理系による通常モードの実行ではランダムに決まるが，トレースモードの実行では**図 2** に示すように指定することができる．

ブレークポイント機能

C や Java 言語の IDE のように特定の行にブレークポイントを設定する方法は，MAC の IDE では不十分である．1 行の中に複数の ambient が並列合成の形で記述されていることも一因だが，MAC のプロセスの持つ非決定的な性質により，そのように設定したブレークポイントを素通りされる可能性があるためである．そのため IDE4MAC では，**図 3**-④に ambient 名（たとえば *SHIP*，*TK*，*co1* な

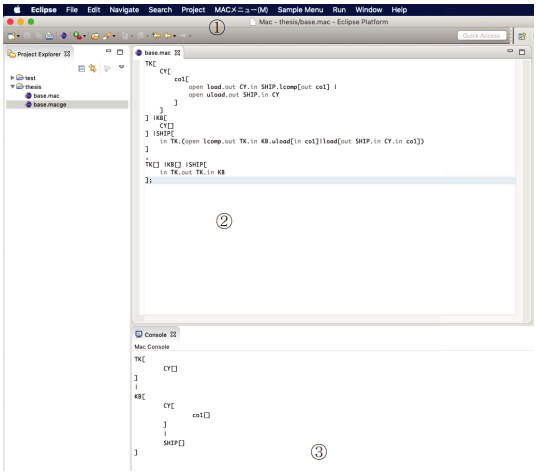


図 1　IDE4MAC のテキストエディタ
Fig. 1　Text editor of IDE4MAC.

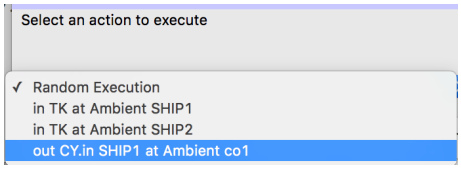


図 2　非決定的動作の選択メニュー
Fig. 2　Menu for non–deterministic choice.

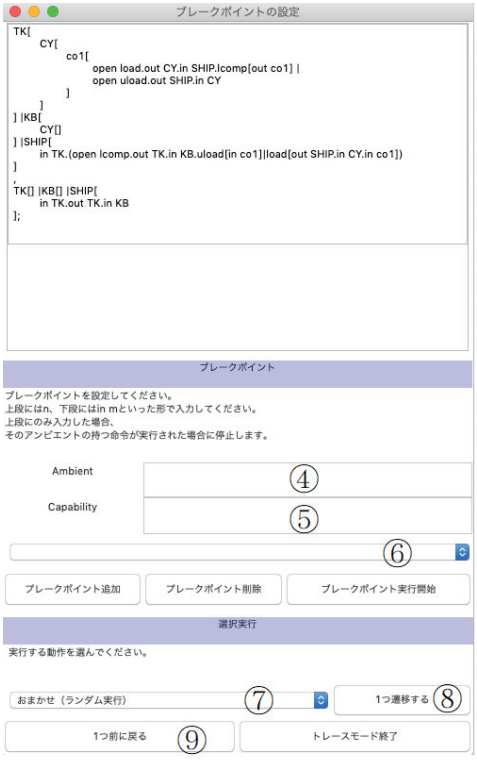


図 3　トレースモード中のポップアップ
Fig. 3　Pop–up window for tracing mode.

[*3] ただし最新の Eclipse では動作せず，現状 Eclipse IDE for Java EE Developers Luna のみで動作を確認している．

ど）を，⑤に capability 名を入力し実行ボタンを押すと，指定された ambient の指定された capability が実行されるとそこで実行を中断するようになっている．

順方向・逆方向ワンステップ実行機能

ブレークポイントで実行が中断しているとき，図 3-⑧のボタンを押すことで，中断中のプロセス内で活性な capability をワンステップずつ実行させることができる．さらに，3-⑨のボタンを押すことで遷移前の状態にワンステップずつ戻すことができる．逆方向のワンステップ実行機能（以降巻き戻し機能）は特に非決定的な動作を目視で確認する作業に有効である．

3.2　グラフィカルエディタ

IDE4MAC の最大の特徴は，**図 4** に示すグラフィカルエディタである．CCS や CSP，π 計算など他のプロセス代数と異なり，AC および MAC は，プロセス式の [括弧] による階層構造がそのまま現実の対象物の構造を表現している．したがって，いかに正確にその階層構造を把握できるかが，MAC プロセスを記述するために重要となる．

テキスト形式のままでも階層構造の理解は不可能ではないが，それをグラフィカルに表現することができれば，直観的に対象物の構造を把握しながら編集することが可能となる．IDE4MAC ではグラフィカルな表現として，階層構造が深くなった場合でもその構造を簡潔に表現できる木構造を選択した．AC や MAC では，プロセス式の遷移にともなう階層構造の変化により対象物の動作を表現しているが，木構造の変化する様子を目視することにより，対象物の動作を直観的に把握することが可能となる．また，ambient や capability を木のノードとして表現することにより，追加と削除，祖先子孫関係にある ambient の位置関係の把握，位置を把握しながらの編集などが容易になる．IDE4MAC の GE ではこのような木構造に対する操作が可能である．

以下，GE の備える各種支援機能について説明する．

図 4-⑪の部分をキャンバスと呼ぶ．この部分にプロセス式の階層構造を表現する木構造を記述する．四角いノードが ambient を，丸いノードが capability を表す．capability も ambient と同様ノードとして表現した理由は，AC において ambient も capablity もそれぞれ独立したプロセスとして扱われるからである．たとえば in A | B[] というプロセス式は，in A.0 というプロセスと B[0] [*4] というプロセスが並行な位置に存在することを表しているため，GE では capability を辺として表すのではなくノードとして表現している．

図 4-⑩の部分をパレットと呼ぶ．ここから ambient や capability をパーツとして取り出しキャンバスに置くことで，木構造を記述することができる．一般的に編集中は図 8 のようにノードが多数現れるが，特定の部分の編集に直接関係しない部分木は理解の邪魔なので，ノードをクリックすることでその下の部分木を縮約・展開することができる．

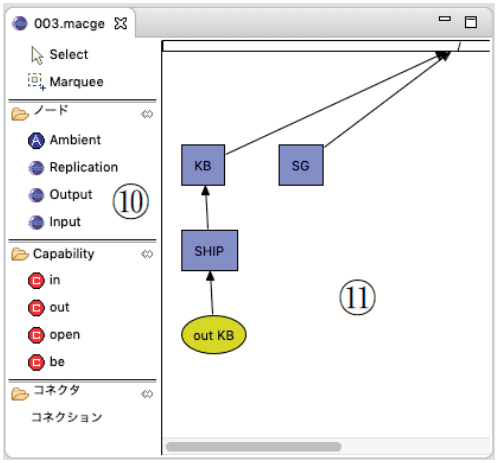


図 4　木構造編集用のグラフィカルエディタ
Fig. 4　Graphical editor for composing tree structures.

GE 上に表示される木は有向辺と無向辺から構成される．有向辺は部分木がどの ambient の子になっているかを表示するために使用され，無向辺は部分木内の capability や ambient の実行順序を示すために使われる．たとえば 4 章の図 12 に示す①の部分木は，有向辺によって co1 ambient の子であることを示しており，その部分木内の無向辺は，ulcomp ambient 内で out co1，out CY，in SHIP_1 の順でそれぞれの capablity が実行されることを示している．

3.3　対応する言語仕様

IDE4MAC では，文献 [1] の構文規則のすべての要素に対応している．ただし名前の制限を表す ν は TE で記述しづらいため，ν の代わりに nu と記述する．たとえば $(nu\ a)(a[\,]|b[in\ a])|a[\,]$ と TE 上では入力する．この場合最初に現れる $a[\,]$ と $b[in\ a]$ の a は束縛名となるため，処理系内では $\#local\_a\#[]$ と変換され，2 番目に現れる $a[\,]$ とは区別される．また GE 上のノードでも各名前が同様に表示される．なお TE ではプロセス定義および定義を利用した再帰的なプロセスの記述および実行が可能であるが，再帰的なプロセス式の GE 上での表現方法に問題があるため，現状では再帰処理を行うプロセスには対応していない．

3.4　TE–GE 連携機能

IDE4MAC では TE と GE を同時に見ながらプロセス式を記述することができる．これを我々は TE–GE 連携機能と呼んでいる．プロセス式を記述する際，一方のエディタ上でファイルが上書き保存された際に，他方のエディタへ変更が自動的に反映される．なお，GE から TE へ連携す

[*4] 0 は何も動作をしないプロセスを表すが，通常は省略して表記される．

る場合は上書き保存以外の操作は不要であるが，TE から GE へ連携する場合には，上書き保存後，Reload ボタンを押す必要がある．

TE–GE 連携機能の動作を説明する．**図 5** 上は TE にプロセス式を記述した状態である．この状態からプロセス式の上書き保存を行い Reload ボタンを押した状態が図 5 下である．プロセス式に構文解析が行われ，木構造に変換後，GE に反映されている．

図 6 上は GE 上の *in AMB_A* の有向辺の接続先を *AMB_B* に変更した状態である．この状態から木構造の上書き保存を行った状態が図 6 下である．木構造の変更が TE に反映されている．

3.5　GE 上のブレークポイント機能

図 7 に示すように，GE 上の任意の capability を右クリックして現れるメニューから「Breakpoint」を選ぶことで，選択した capability を消費するまで MAC の遷移規則に基づいた木構造上での遷移が行われる．

その結果は TE–GE 連携機能により両エディタに即時反映される．先に示した図 4 は，この遷移後の木構造を表したものである．TE と同様，GE 上での遷移も任意の状態まで巻き戻すことができる．

3.6　GE 上の個別式切り替え機能

MAC が AC や他のプロセス代数と大きく異なるのは，2.2 節の式 (3) で示したように，個々の対象物の性質を個別式で表現し，それらの組合せで全体の動作を表現している点である．個別式それぞれの同期的な動作を組み合わせて全体の動作の同期，たとえばすべてのコンテナが積み込まれた後に船の出港が許可されるといった動作が表現される．したがって，個々の個別式それぞれの同期的な動作を見比べながらプロセス式を作成する必要がある．

TE では個別式をコンマで区切って一括して表示するが，GE ではこの区切りを解析し，個別式をそれぞれ独立した木構造として表示する．**図 8** は，式 (3) の 2 行目に示したコンテナ *co*1 の個別式（の詳細版）を表す木構造である[*5]．

この図の⑫の部分で，全体式中に存在する任意の個別式を表示することができる．⑫の部分で world6 をクリックすると**図 9** の表示に切り替わるが，これは式 (3) の 5 行目に示した *SHIP* の個別式に相当する木構造である[*6]．

GE の持つこの機能により，各個別式の木構造を表示しそれぞれの構造を把握しながら，対象となるシステム全体をモデル化することが可能となる．

[*5] 個別式はそれぞれ独立した世界を表現していると見ることもできるため，GE では各個別式に world1，world2，··· と名付けて識別している．

[*6] この例では，式 (3) にコンテナを 3 個追加した式で個別式が 6 個存在する．

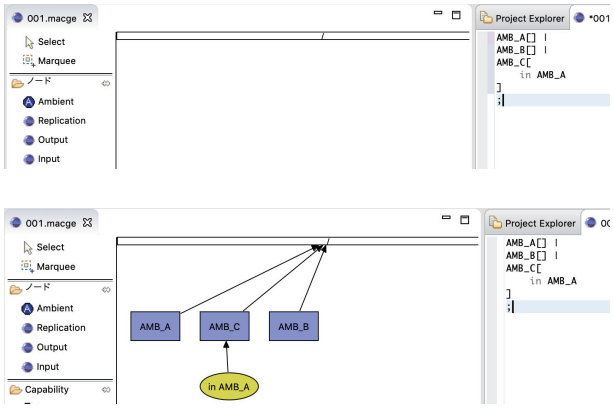


図 5　TE でのプロセス式記述と GE への反映
Fig. 5　Composing formulae on TE and refrection to GE.

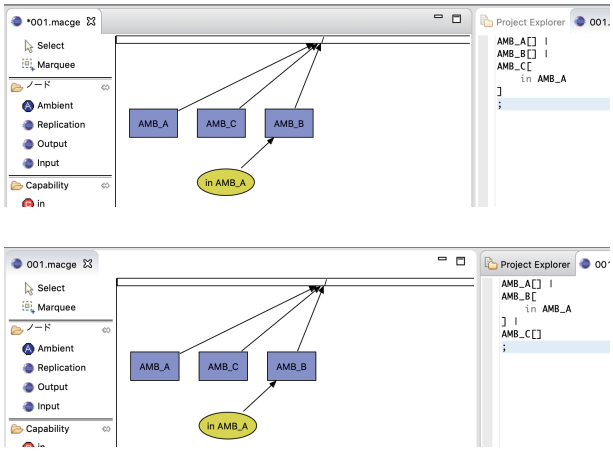


図 6　*in AMB_A* の有向辺接続先変更
Fig. 6　Changing the directed edge of *in AMB_A*.

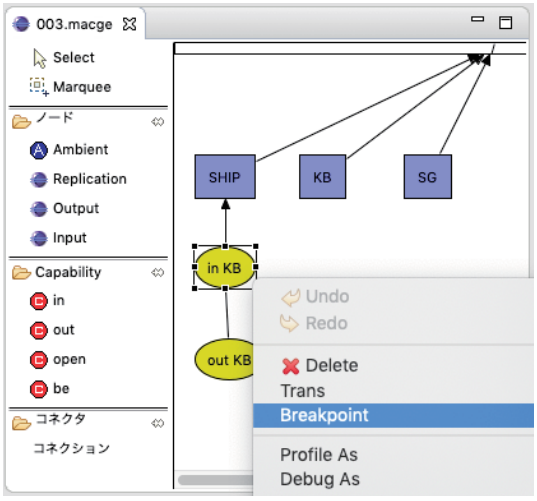


図 7　GE でのブレークポイント指定の様子
Fig. 7　Setting a breakpoint on GE.

3.7　大域 ambient 検索機能

2.2 節に示したように，MAC では船などの同期制御（すべてのコンテナが積まれてから船は出港可能になるなど）を，各個別式に現れるすべての同じ名前を持つ大域 ambient（*SHIP* など）が遷移可能になったかどうかで表現する．そ

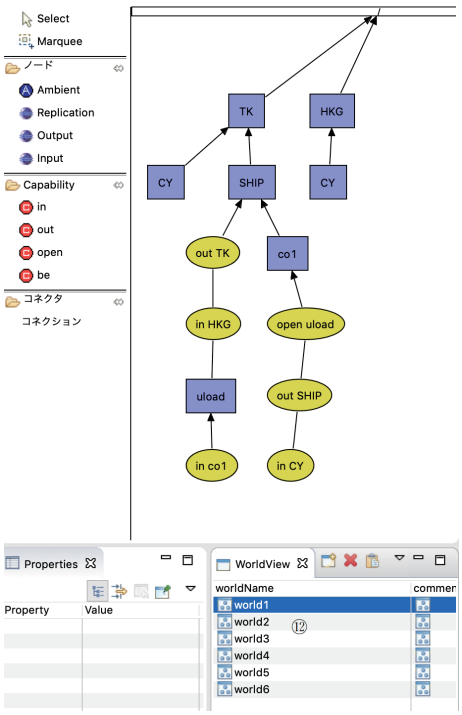


図 8　*co*1 の個別式

Fig. 8　Indivisual expression of *co*1.

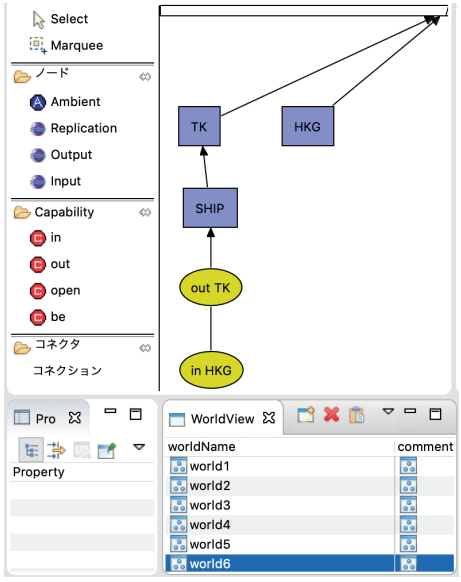


図 9　*SHIP* の個別式

Fig. 9　Indivisual expression of *SHIP*.

のため，各個別式に現れる共通する大域 ambient の特定が重要となる．これに対し IDE4MAC は，GE 上で ambient を指定することで，TE 上で同じ名前を持つすべての ambient を強調表示する機能を備えている．これにより，同期制御に問題がある場合にその場所の特定が容易になる．**図 10** は，GE 上で赤丸で囲んだ SHIP ambient のノードを指定し，TE 上ですべての SHIP ambient が強調表示されている様子を表している．

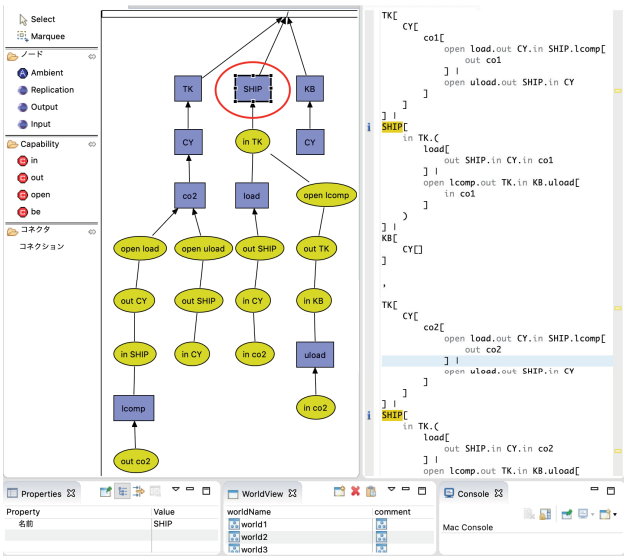


図 10　大域 ambient 検索機能

Fig. 10　Searching function for global ambients.

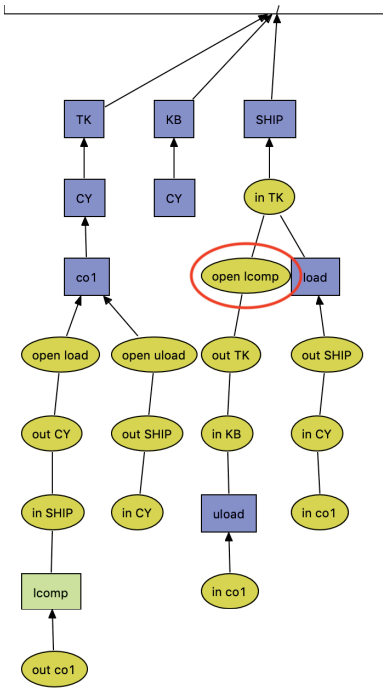


図 11　capablity–ambient の関連検索機能

Fig. 11　Searching function for the relation of capability and ambients.

3.8　capablity–ambient 関連検索機能

木の規模が大きくなった場合，capability とそれが作用する離れた位置にある ambient の対応関係を目視で認識することが困難になる．これに対し IDE4MAC は GE 上で，ある capability ノードを指定することで，それが作用する可能性のある ambient を強調表示する機能を備えている．この機能は，特に 4 章で述べるような，同じ ambient に作用する複数の capability の競合が生じている場合，その位置を把握するために有効である．**図 11** は，GE 上で赤丸で囲んだ *open lcomp* capability のノードを指定し，離れた

位置にある *lcomp* ambient が強調表示されている様子を表している．

3.9　MAC の処理系

IDE4MAC では，我々が物流監視システムのために開発した MAC の処理系を利用してプロセス式が実行される．エディタで記述したプロセス式を実行する際，まずプロセス式がこの処理系に渡される．この処理系により，MAC のプロセス式は構文解析され Java のオブジェクトに変換され，MAC のプロセス式の木構造を表現したオブジェクトの木構造として Java 仮想機械（JVM）上で管理される．

各オブジェクトは capability に対応するメソッドを持っており，そのメソッドを実行することでオブジェクトの木構造を変化させ，MAC のプロセス式の遷移を実現している．実行後，オブジェクトの木構造からテキスト形式の MAC プロセス式の木構造を生成し，エディタに再表示される．グラフィカルエディタで記載された木構造は，いったんプロセス式に変換されたうえでこのように処理される．

4.　プロセス式デバッグ例

1 章で述べたように，我々が研究している物流監視システムでは，MAC のプロセス式で物流システムの動作を記述し，そのプロセス式を実際のコンテナの取扱の監視に利用している．IDE4MAC には PAT [10] のようなモデル検査機能はないが，これまでに述べてきた機能を用いて，開発途中のプロセス式に対するデバッグを行った実例について述べる．

実際の物流システムでは様々な作業が並行して行われ，それらの間の同期が重要となる．下記はその一例である．

条件 1：積降ろしにおける船の出港制限

船は積降ろし港において，予定されたすべてのコンテナの積降ろしが完了するまで出港してはならない．

条件 2：積込みにおける船の出港制限

船は予定されたすべてのコンテナの積込みが完了するまで出港してはならない．

これらに反する動作が起こった場合警告を出すことが物流監視システムの役割の 1 つである．条件 1，2 はそれぞれ式 (4)，(5) で表すことができると考えていた．これらの式に現れる *SG* はシンガポール港を表している．

$$
\begin{aligned}
&SG[\\
&\ SHIP\_1[open\ ulcomp.out\ SG.(\text{中略})]\\
&\ |CY[co1[ulcomp[out\ co1.out\ CY.in\ SHIP\_1]]]\\
&]
\end{aligned}
\tag{4}
$$

式 (4) は，*SHIP_1* で運ばれてきたコンテナ *co1* が *SG* 港で積み降ろされ，*SG* 港のコンテナヤード *CY* に搬入された状態を表している．この後，*co1* 中の制御用 ambient *ulcomp* が *co1* および *CY* を出て *SHIP_1* に入り，*SHIP_1* 中の *open ulcomp* が消費されることによって *SHIP_1* 中の *out SG* が活性になり，*SHIP_1* が *SG* 港を出てもよいことを表現している．つまり制御用 ambient *ulcomp* が *SHIP_1* に対して，*co1* の積降ろしが完了したことを通知しているのである．

$$
\begin{aligned}
&SG[\\
&\ SHIP\_2[open\ lcomp.out\ SG.(\text{中略})]\\
&\ |CY[co1[out\ CY.in\ SHIP\_2.lcomp[out\ co1]]]\\
&(\text{以下略})\\
&]
\end{aligned}
\tag{5}
$$

式 (5) は，*SG* 港のコンテナヤード *CY* にあるコンテナ *co1* を今から *SHIP_2* に積み込もうとしている状態を表している．まず *co1* が *CY* を出て *SHIP_2* に入る．その後 *co1* 中の制御用 ambient *lcomp* が *co1* から出ることによって *SHIP_2* 中の *open lcomp* が消費され，*SHIP_2* 中の *out SG* が活性になり，*SHIP_2* が *SG* 港を出てもよいことを表現している．つまり制御用 ambient *lcomp* が *SHIP_2* に対して，*co1* の積込みが完了したことを通知しているのである．

式 (4)，(5) はそれぞれ単独では問題なく動作し，条件 1，2 を適切に表現できているように思われていた．しかし IDE4MAC 上でこれら 2 つの式を合わせて *co1* の個別式として作成し動作させた場合，このとおりに動く場合と，*SHIP_1* が出港せず，*SHIP_1* に載せられている他のコンテナが目的地に到着しない場合があることが分かった．原因は *co1* 内の *ulcomp* が *co1* を出る前に *co1* が *SHIP_2* に移動してしまい，*ulcomp* が *SHIP_1* に移動できなかったためである．これは AC の非決定性に起因する同期の失敗例であり，grave interference と呼ばれている AC 特有の落とし穴であった [7]．文献 [13] で構築したモデル検査システムでももちろん反例としてこの現象はとらえることができるが，そのデバッグは困難であった．モデル検査システムでは，コンテナが目的港に到着せず途中で停止し，それ以上どのような遷移もできずに実行が終了した状態のプロセス式が反例として表示される．しかし，表示されるテキスト形式のプロセス式から目視でその原因を特定することはきわめて煩雑な作業であった．

これに対し IDE4MAC の TE–GE 連携機能を用い，**図 12** に示すようにこの状態の式と木構造を見ながらブレークポイント機能と巻き戻し機能を使うことで，正しく動作する場合と grave interference が起きる場合の式の構造や実行順序を確認することができた．この図の①が条件 1 を表した式 (4)，②が条件 2 を表した式 (5) の該当部分である．

この図を見ることで，*co1* 内にはこの ambient に作用する *out CY* という capability と，*co1* から外に出ようとし

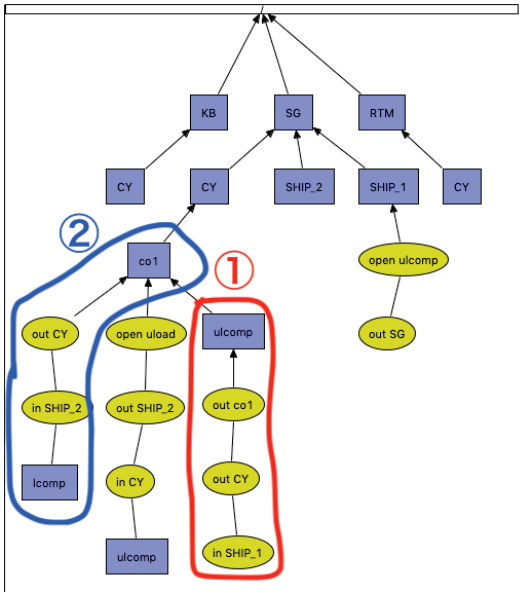


図 12　grave interference 発生場所の特定
Fig. 12　The place where a grave interference occured.

ている *ulcomp* ambient が並行な位置にあることが目視でき，*out CY* と *ulcomp* の *out co1* の実行順序によっては，*ulcomp* のその後の遷移に影響が出ることが分かる．

一般的に，意図しない状態で遷移が停止した場合の原因の確認方法は以下のとおりである：

1) ランダム実行モードで遷移させ，意図しない状態で遷移が終了してしまったプロセス式を GE 上で表示させる．
2) 巻き戻し機能を用いてその直前の状態までプロセス式の木を巻き戻し，各ノードの位置関係を確認し，意図どおりの位置関係になっていなければさらにもう 1 歩巻き戻す．これを，各ノードの位置関係が意図どおりになっている状態（たとえば図 12 の状態）まで繰り返す．
3) 上記の状態に達した際に，1 歩後の状態に至る際に使用された capability にブレークポイントを設定する．
4) そのブレークポイント以降の様々な非決定的な動作を何度も行うことで，意図どおりではない遷移の様子を目視で確認できる．

以上のように，GE を用いて原因の特定を比較的簡単に行うことができた．

1) についてはモデル検査システムでもテキスト形式のプロセス式に対してなら可能であるが，そこからの巻き戻しや，木構造での各 ambient や capablity の目視は不可能であり，GE の各機能の有効性が確認できたと考える．

さらに，ブレークポイント機能と巻き戻し機能を使うことで *ulcomp* の移動を *co1* の移動よりも優先させるように式を修正することが容易にでき，意図しない状態で停止してしまうような実行系列を排除することが可能となった．

もちろんランダム実行モードを用いて，プロセス式が意図しない状態で停止しないことを確認したとしても，非決定的選択によって生じるすべての実行系列を網羅したことにはならない．それを保証するのはモデル検査器の役割である．IDE4MAC のデバッガの役割は，不具合が残っている場合その修正作業を支援することである．

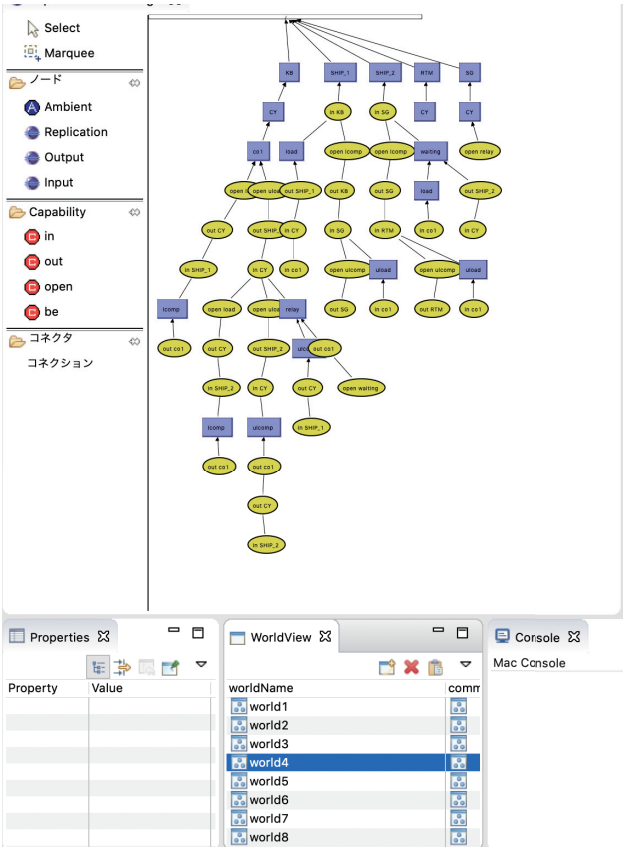


図 13　大規模なモデル化の例
Fig. 13　Example of large scale modeling.

文献 [13] で構築したモデル検査システムは，特定の 2 つの時相論理式のみに特化した検査を行うものであり，統合開発環境に組み込めるような汎用性は現在の所持っていない．将来的にはモデル検査システムの汎用性を高めて IDE4MAC と連携させ，1) で示したようなプロセス式の検出をモデル検査システムで行い，そこから 2) 以降の手順を実行できるように実装することが有効と考えている．

5.　大規模なプロセス式での動作確認

IDE4MAC が大規模なモデル作成にも耐えられることを示すため，コンテナ 100 個，船 3 隻，港 4 つを使用する物流計画を表すプロセス式（ノード数 6151）の作成および実行を行った．このプロセス式は，神戸発シンガポール行の *SHIP_1*，シンガポール発ロッテルダム行の *SHIP_2*，ロッテルダム発ハンブルク行の *SHIP_3* を用いてコンテナ 30 個を神戸からロッテルダムへ，コンテナ 20 個を神戸からハンブルクへ，コンテナ 50 個をシンガポールからハンブルクへ輸送するという運送計画を表したものである．**図 13** は，

co1（コンテナ 1）の個別式をモデル化した world4 を選択表示した様子を示している．world1～world3 が *SHIP_1*～*SHIP_3*，world4～world103 が *co1*～*co100* の個別式の GE での表示名である．

実験環境は下記のとおりである．

モデル：MacBook Pro（13-inch，2017 年モデル）
プロセッサ：2.3 GHz Intel Core i5
メモリ：8 GB 2133 MHz LPDDR3
OS: MacOS Mojave 10.14.6（18G2022）

この環境において，式や木構造の記述，相互変換などはタイムラグなく操作可能であり，すべてのコンテナが仕出港から仕向港に達し遷移が完了するまでにかかった時間は 4 分 35 秒であった．その際の遷移数は 3,772 回であった．

6. 関連研究

6.1 PAT

プロセス代数に対する統合開発環境としては CSP [6] に対する PAT [10], [12] が有名である．PAT は単なる統合開発環境ではなく，様相論理を用いて記述された性質をプロセス式が満たすかどうかを確認するためのモデル検査機能を有する統合検証環境であり，PAT を用いたモデル検査に関する成果が多数報告されている．たとえば文献 [11], [12] では，ネットワーク上に分散されたノードから代表元を選出する，いわゆるリーダ選出プロトコルを PAT で実装し，実装されたプロセスがつねにただ 1 つのリーダを選出できるかどうかを PAT で検証できることを示している．また文献 [19] では，UML のシーケンス図で表されたシステムを CSP に変換して PAT を用いてそのシステムに対するモデル検査を行うためのツールが提案されている．PAT に付属するチュートリアル [10] 内では，文献 [14] で提示されている列車運行管理システムの安全性を検証する例を PAT で記述し，PAT でもその安全性が検証できることを示している．

これらの研究では，CSP で記述されたモデルに対し PAT でモデル検査を実施し，必要であれば反例を示して，モデル内部で所期の性質を満たさない部分を特定できることを示している．しかしモデルそのものを生成したり修正したりすることをいかに PAT が支援しているかについては述べられていない．PAT にはプロセス式と状態遷移図を見比べる機能はあるが，プロセス式の構造をグラフィカルに表示し作成を支援する機能は持たない．

これに対し IDE4MAC は，構造理解が重要な MAC のプロセス式に対し，TE と GE の連携により，4 章で述べたように複雑な同期制御を持つ MAC のプロセス式の作成に有効であると思われる．また 3.6 節で述べたように，MAC のプロセス式は他のプロセス代数と違い複数のコンポーネント（個別式）から構成されるため，各コンポーネントの木構造の表示を切り替えながらプロセス式の同期制御を設計できることも IDE4MAC の特徴となっている．

6.2 DDD

DDD（Data Display Debugger）[2] は，IDE4MAC と同様デバッグ中にデータ構造を可視化し，その変化を確認できる機能を有する．たとえば構造体をノードとするグラフや木の構造を視覚的に理解し，各ノードの持つ値の変化を追っていくことが可能である．しかし，基本的に 1 度表示された木構造内のノードの位置は変化しないため，ノード間の接続関係が変化するような実行，すなわち木構造の動的な変化を追っていくことには難点がある．そのような場合ノードの位置は変化せず，リンクだけがつけ代わって表示されるため，変化後の関係が理解しにくい．

これに対し AC や MAC では，ノード（つまり ambient）間の親子関係が変化することがプログラム実行の本質であり，IDE4MAC では in や out capability の実行のたびにノードを再配置して表示するようになっている．これにより，現実世界の対象物（たとえば物流システムにおける船やコンテナ）の位置関係の動的な変化が理解しやすくなっている．

また，DDD には Backtrace という名前の機能はあるが，これは関数呼び出しスタックの状態を表示するものであり，IDE4MAC のようにブレークポイントからワンステップずつ実行状態を巻き戻し，そこからまた前方向にステップ実行できる機能ではない．このため非決定的な選択を持つようなプログラムのデバッグには適さないと考えられる．

6.3 BlockEditor

文献 [8] では，Java プログラムの構造（メソッド，while 文，if 文など）をブロック図を用いて可視化する BlockEditor が提案されている．テキスト形式のプログラムとブロック図の相互変換や，ブロックの内部を隠して折りたたみ全体の構造を見やすくしたり，内部を展開してブロックの詳細を確認できるなどの展開畳み込み機能など，IDE4MAC と同様の機能を有するが，デバッグ機能は持たないため，実行時のデータの変化を追跡することはできない．4 章に示したように，MAC プロセス式の持つ動的な階層構造という特徴を木構造を用いてビジュアルに表示し，その状態でトレース，バックトレース，そしてノード内部の編集ができる IDE4MAC の機能が MAC プロセス開発には特に有効であると考えている．

6.4 Reversible Debugging

ブレークポイントにおける一時停止時やステップ実行，巻き戻し機能は GDB や商用のデバッガで実用化されているが，Eclipse や Visual Studio など一般的な統合開発環境では巻き戻し機能はサポートされていない．また GDB や商用のデバッガではプログラムの可視化はサポートしてお

らず，データ構造の動的な変化をグラフィカルに確認しながら後戻りすることはできない．AC や MAC プロセスの持つ非決定的な性質に対応するためにも IDE4MAC の持つこの機能は特に重要である．

7. 結論

本稿では，多重 Ambient Calculus（MAC）のための統合開発環境である IDE4MAC を提案し，IDE4MAC の持つ物流システムのモデル化に有用な各種機能について説明した．物流システムには，動的に階層構造が変化し（船へのコンテナの積込みや船の入出港など），さらに構造の構成要素が動的に増減する（コンテナの追加・キャンセルなど）という特徴がある．MAC はこの特徴をモデル化するために設計された言語である．他のプロセス代数と異なるこの特徴により困難であったプロセス式のデバッグに対し，IDE4MAC の各種機能が有効に機能することを示した．特に，AC に固有の同期的な動作の失敗原因である grave interference によって生じていたバグの特定および修正が，IDE4MAC の持つグラフィカルな巻き戻し機能によって可能だったことを示した．プログラムをグラフィカルに表示する機能，およびバックトレース機能については DDD，BlockEdior，GDB などが C や Java に対し実現しているが，それらはグラフィカル表示またはバックトレースのどちらかの機能しか有しておらず，IDE4MAC はその両方を備えることで MAC プロセスの持つ非決定的な性質に対応したデバッグが可能であることを示した．

IDE4MAC はもちろん AC のプロセス式作成にも有効である．我々は現在，強化学習エンジン fineOptimAI を用いた組合せ最適化の研究を行っている [18]．このなかで，我々が強化学習の対象としている仕入れ最適化やコンテナ荷役効率化の問題に対し，IDE4MAC を用いて作成したモデル（AC のプロセス式）を活用している．これらの問題を AC でモデル化し，AC の持つ非決定的な性質を用いて実行系列を網羅的に書き出し，その実行系列を中心に学習をすすめることで，fineOptimAI の性能を向上させることが可能となる．このように IDE4MAC は物流監視システムだけでなく，階層構造が動的に変化する様々な対象に関する問題に応用することが可能である．

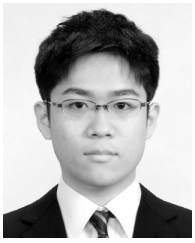
謝辞　本研究は，JST START（JPMJST1914）の支援を受けたものである．

参考文献

[1] Cardelli, L. and Gordon, A.D.: Mobile Ambients, *Theoretical Computer Science*, Vol.240, pp.177–213 (2000).
[2] GNU プロジェクト：DDD–Data Display Debugger (2003), 入手先 ⟨https://www.gnu.org/software/ddd⟩.
[3] 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, pp.1–12 (2013).
[4] 樋口昌宏，森田哲平，加藤　暢：多重 Ambient Calculus による物流記述に対する弱双模倣等価性を用いたモデル検査，情報処理学会論文誌：プログラミング，Vol.5, No.3, pp.56–60 (2012).
[5] 樋口昌宏，加藤　暢：物流システム記述のための多重 Ambient Calculus，情報処理学会論文誌：プログラミング，Vol.5, No.2, pp.79–87 (2012).
[6] Hoare, C.: *Communicating Sequential Processes*, Prentice Hall (1985).
[7] Levi, F. and Sangiorgi, D.: Controlling Interference in Ambients, *Proc. 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp.352–364, ACM (2000).
[8] 松澤芳昭，保井　元，杉浦　学，酒井三四郎：ビジュアル–Java 相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価，情報処理学会論文誌，Vol.55, No.2, pp.57–71 (2014).
[9] 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，Vol.48, No.SIG 10(PRO33), pp.151–164 (2007).
[10] project, P.: PAT: process Analysis Toolkit, available from ⟨https://pat.comp.nus.edu.sg⟩.
[11] Si, Y., Sun, J., Liu, Y., Dong, J.S., Pang, J., Zhang, S.J. and Yang, X.: Model checking with fairness assumptions using PAT, *Frontiers of Computer Science*, Vol.8, pp.1–16 (2013).
[12] Sun, J., Liu, Y., Dong, J.S. and Pang, J.: PAT: Towards Flexible Verification under Fairness, *Proc. 21st International Conference on Computer Aided Verification* (*CAV'09*), LNCS, Vol.5643, pp.709–714, Springer (2009).
[13] 加藤　暢，高岡久裕，樋口昌宏，大山博史：多重 Ambient Calculus を用いた動的な海上物流計画に対するモデル検査，情報処理学会論文誌：数理モデル化と応用，Vol.11, No.3, pp.84–99 (2018).
[14] Yi, W., Pettersson, P. and Daniels, M.: Automatic Verification of Real–Time Communicating Systems by Constraint Solving, *Proc. 7th International Conference on Formal Description Techniques*, North-Holland, pp.223–238 (1994).
[15] 国土交通省：メコン地域陸路実用化実証走行試験 (2007), 入手先 ⟨http://www.mlit.go.jp/kisha/kisha07/15/151018/01.pdf⟩.
[16] 国土交通省：国土技術政策総合研究所資料（海上輸送を中心とした最近のサプライチェインセキュリティの動向（その 2））(2010)，入手先 ⟨www.nilim.go.jp/lab/bcg/siryou/tnn/tnn0585pdf/ks0585.pdf⟩.
[17] 国土交通省：コンテナ物流情報サービスシステム「Colins」の中国との連携について (2011)，入手先 ⟨www.mlit.go.jp/report/press/port02_hh_000053.html⟩.
[18] 平嶋洋一，加藤　暢，青山正人，黒田規義：組合せ爆発を計算可能な小さな AI「fineOptimAI（ファインオプティマイ）」の事業化，科学技術振興機構大学発産業創出プログラム START (2019)，入手先 ⟨https://www.jst.go.jp/start/project/index.html⟩.
[19] 海津智宏，磯部祥尚，鈴木正人：SDVerifier：プロセス代数 CSP を用いたシーケンス図検証ツール，コンピュータソフトウェア，Vol.32, No.1, pp.1_234–1_252 (2015).
[20] 東芝ロジスティクス：東芝ロジが異業種向けに次世代グローバル混載サービス拡大，カーゴニュース，10 月 13 日 (2015).
[21] 経済産業省：物流業界における電子タグ等の活用実証実験国際コンテナ輸送 (2005)，入手先 ⟨www.mlit.go.jp/kisha/kisha05/10/100329_2/02.pdf⟩.

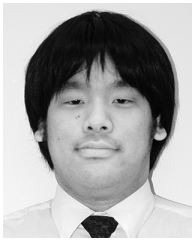
加藤 暢　（正会員）

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，プロセス代数に関する研究に従事．



山田 瞭太　（学生会員）

2020 年近畿大学理工学部情報学科卒業．学士（工学）．現在，近畿大学大学院博士前期課程在籍．プロセス代数に関する研究に従事．



鳥山 颯斗

現在，近畿大学理工学部情報学科在籍．プロセス代数に関する研究に従事．

大倉 亮介

現在，近畿大学理工学部情報学科在籍．プロセス代数に関する研究に従事．

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．