ソフトウェア工学7－3
（1978. 7. 21）

ACTOR理論と分散型処理系

米澤明憲（東京工業大学 情報科学科）

§1§ はじめに

本論文は，MITのLaboratory for Computer Science（略してLCS，Project MACの新称）の研究プロジェクトのひとつであるMessage Passing Semantics Group（C. Hewittが主宰者）において，理論・応用両面で様々な角度から研究されているACTOR理論を，分散型処理系の挙動に対する基礎的理論とみなす観点から紹介・考察するものである。

ACTOR理論は，C. Hewittが，彼自身の提案した人工知能研究における，向題解決用言語PLANNER[13]のパターン・マッチング機能のセマンティクスを説明するために用いた"メッセイジ・パッシング"によるメタフォーが，きっかけとなって発展したもので，ここで考察する並列処理・分散処理のための計算モデルとしての役割の他に，人工知能研究における"知識"の表現・格納・使用のための一般的形式を与えようとする研究[14][15]の所産でもある。

以下，次節・第3節においてACTOR計算モデルを説明し，第4節で，分散型処理系の機能的側面に対する基礎的モデルとしての諸性質を考察する。第5節では，分散型処理系，またより一般的なマルチプロセス系を扱うのに有効な，筆者による仕様・検証技法について簡単に触れ，第6節でACTOR理論に基礎を置くその他の研究について若干の説明を加える。

§2§ ACTORとは？

ここで与える計算モデル[15][16]は，actorと呼ばれるオブジェクトと，そのオブジェクト同志の間で行われる"メッセイジのやりとり"（message passing）とに基づく。actorとは，プログラミング言語の概念を用いて直感的に述べれば，手続（procedure）や関数（function）などの計算や操作を行う範ちゅうに属するものと，記憶場所，データ，データ構造などの情報の格納のための範ちゅうに属するものという，ふたつの独立した概念を，メッセイジを受けとることによって能動化（active）される手続的（procedural）なオブジェクトとして統一したものである。メッセイジは，actorにおけるオペレイションの要求（request）や，そのオペレイションを遂行するのに必要な情報などを含む場合と，要求に対する返答（reply）やオペレイションの結果を含む場合がある。（図1参照）

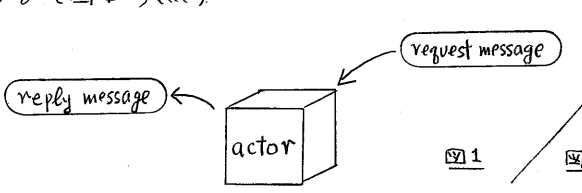

図1

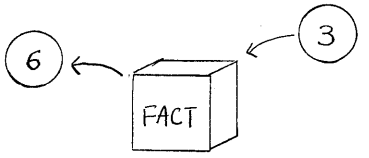

図2

たとえば，階乗を計算して答を返すactorをFACTとすると，FACTは3というメッセイジ（又は要求）を受けとると，6を結果として出す（図2）。一般に，計算や操作を行うactorに送られるメッセイジには，手続や関数呼び出しに使われる実引数に対応するものが含まれていると考えてよい。（図4，次頁）

アレイやディメンジョンに相当するactorをAとすると，i番目の要素を取り出すには，Aに例えば(element: i)という要求を含むメッセイジを送ればよい。i番目の要素を新たな要素new-elementで置換えるには，例えば(update: i new-element)という形の要求を含むメッセイジを送ればよい。(図3)

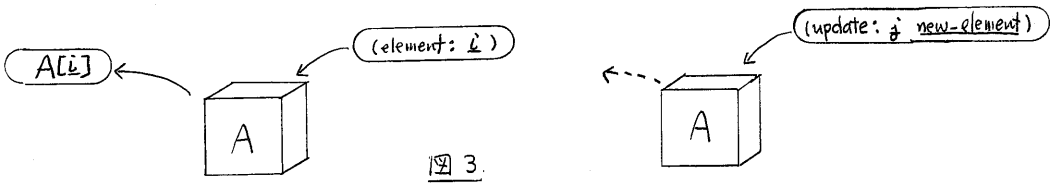

図3

簡単なデータ構造のひとつであるプッシュ・ダウン・スタックに対応するactorに対しては，(pop:)という要求で現在のスタックの先頭要素を取り除き，(push: new-element)で新しい先頭要素を載せることができる。一般に，データ構造のようにふるまうactorに送られるメッセイジには，格納されている情報を読んだり換えたりするためのオペレイションを指示するものと，オペレイションに必要なデータを含む。(図4)

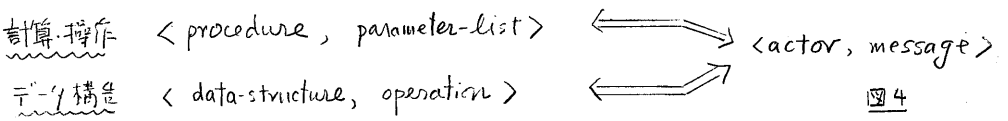

図4

上の直感的な説明から明らかと思われるが，actorという概念を用いて，手続や関数，またプロセス，プロセッサやコンピュータ網に接続されたコンピュータシステムまでモデル化できる。さらに，ハードウェアのメモリ，メモリバンク，ソフトウェアよでの種々のデータ構造のみならず，ファイル，ファイルシステム，データベース，そのマネイジメント・システム等も，必要に応じて，単一のactorとしてもまた，何らかの構造をもったactorの集団としても，見通しよくモデル化できる。以下，このようなactorの概念について，もう少し詳しく述べてみることにする。

§3§ ACTOR計算モデル

3.1 actor

actorは概念的にふたつの部分：scriptとacquaintancesにより構成される。scriptとは，受けとったメッセイジに対して，そのactorが反応して，どのようにふるまうかを記述したものである。(手続的に記述されて考えてよい。) 各actorは各々ある固定したメッセイジの(型の)集合をもち，その集合に属するメッセイジによって能動化され，scriptに従ってふるまう。その集合に属さないメッセイジに対する反応は，未定義とする。actorのacquaintancesとは，そのactorが知っている(knows-about)のactorの有限集合で，あるactor Aがactor Bを知っている時のみ，AはBへメッセイジを直接送ることができる。(acquaintancesの要素がどのように決められるかについては，4.3で述べる。)

actorは，はじめから存在しているもの中，"計算"の過程で，他のactorによって生成されることもある。"計算"(computation)を定義するために，次に説明するイベント(event)という概念が必要となる。

3.2 イベント (event) と継続 (continuation)

メッセイジ*) M の actor T への到着をイベントと定義し，次のような記法で表わすことがある。（数学的には，T と M の対として定義される。）

$$[\![\, T \Leftarrow M \,]\!]$$

このとき，T をこのイベントの標的 (target) と呼び M をイベントのメッセイジと呼ぶ。イベントはメッセイジの純粋な到着のみを意味し，メッセイジによって要求されたオペレイションの遂行や完了を意味しない。

メッセイジの中には，要求するオペレイションを示すものや，オペレイションに必要なデータの他に，オペレイションの結果を送る，送り先に相当する actor を含むことがある。このような actor を継続 (continuation) と呼ぶ。継続を含んだメッセイジの一般形は，次のような記法で表わすことがある。

[request: request reply-to: continuation]

継続 C を含むメッセイジ M が，actor T に到着したとすると，M によって要求されたオペレイションの結果 R は，最終的に C に到着する（図5）。

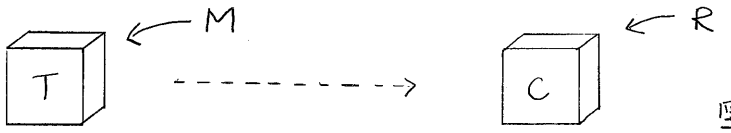

図5

3.3 計算 (computation)

actor モデルにおける計算 (computation) は，イベントの半順序集合として定義される。順序づけは，狭義 (strict: 同一元以外は決して同順位でない) で，時間的 "先行" (precedes) に対応し "---→" で表わす。actor の集団の中で，メッセイジの伝送は同時にふたつ以上行われてもよい（図6）。この場合が並列 (parallel) 計算に対応する。3つのイベント E_1, E_2, E_3 に対して，$E_1 \dashrightarrow E_2$ かつ $E_1 \dashrightarrow E_3$ であるが，E_2 と E_3 の間に順序づけがないとき，E_2 と E_3 は並列的 (concurrent) におこったと考えてよい。半順序集合の特別な場合として，同時に2つ以上のメッセイジの伝送が許されないものが，順次的 (serial) 計算に対応し，イベントの全順序集合である（図7）

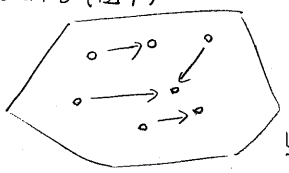
図6

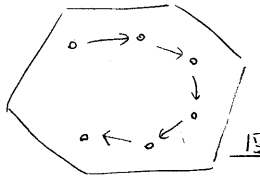
図7

順序づけの狭義性は，どのようなイベントも，自分自身に先行できないことを保証し，デッドロックや starvation のないことを検証する際に有用な性質でもある。さらに，互に順序づけられた任意の2つのイベントの間には，有限個のイベントしか存在しないこと，各イベントに対して，その直後におこるイベントは常に有限個しかないという仮定を設けることにより，計算の物理的な実現可能性を，保証する。

---

*) メッセイジも ひとつの actor であると見なして，理論に何ら矛盾を生じない。

3.4 activation ordering と arrival ordering

上述の時間的"先行"に対応する順序づけは，2種類の順序づけ：activation ordering と arrival ordering の合併とみなされる。activation ordering は，物理的な因果関係を表現するためのもので，"$-act\rightarrow$" という記法が用いられる。$E_1 - act \rightarrow E_2$ が成立するとき，E_2 は E_1 によって引き起された (caused) ものと考える。1つのイベントによって，2つ以上のイベントが，互に順序づけなく引き起されてもよい。(fork に対応。) arrival ordering は1つの actor A に対して，2つ以上のメッセイジが送られたとき，その到着の順序によって定まる順序づけで，"$-arr\rightarrow_A$" で表わされる。この場合，"同時"ということがなく，必ず線型に順序が入られると仮定する。E_1 と E_2 がともに actor A を標的にもつイベントとすると，次のどちらか，$E_1 - arr\rightarrow_A E_2$ か $E_2 - arr\rightarrow_A E_1$ が成立する。arrival ordering は各 actor にそれぞれ定義される。時間的"先行"の順序づけにおいて仮定された，順序の現象と，2つの有限性の条件は，activation ordering とともに arrival ordering についても，勿論満されなければならない。

§4§ 分散型処理系のモデルとしての ACTOR 理論

メッセイジを受けることによって，ある定められた挙動を示すオブジェクトである actor は，本質的に自律的なオブジェクトであるプロセスが，プロセッサをモデルするばかりでなく，第2節で例示したように，情報格納という機能をもつ，純粋にデータ的なオブジェクトである メモリ，バッファ，メモリ・バンク，ファイル などもモデル化する。また，概念的に大量のデータの上に，それを管理・使用するためのソフトウェアを被せたものと考えられる，データ・ベース・システムや，その構成要素も actor によってモデル化できる。それ故，こうしたプロセス，プロセッサ，メモリ，データ・ベースなどを，種々の緊密度と位相で結合した計算機システムが，互にメッセイジを伝送し合う actor の集団として表現できることは明らかであろう。

4.1 分散型処理系

分散型処理系 (distributed system) の定義は多くの研究者によって提案されている[6][22]が，ここでは，多数のプロセッサと記憶装置が，何らかの方式で物理的に結合されたもので，システム全体を制御する中枢的部分が，ハードウェア的にも，ソフトウェア的にも存在しないものと定義する*。オペレイティング・システム及びデータ・ベースの非集中度や，実際のハードウェアの地理的な分散度などにより，ARPA net 等の大規模な計算機網から，10^5 以上のマイクロプロセッサを結合したシステム[30]まで，広いスペクトラムのマルチ・コンピュータ・システムも，この定義で把えられる。

上の定義からも合意されるが，分散型処理系を原理的に特徴づける最も重要な性質は，システムのいかなる構成要素も，その時点でのシステム全体についての正確な情報を持ち得ないということである。メッセイジ伝送のための物理的な"遅れ"が無視できないことや，1つの構成要素が，他の多数の構成要素の，時間的に変わってゆく状態をモニタする煩雑さのために，常に局所的 (local) な情報のみに基づいて，システムの各部が動作しなければならない。これは，集中型処理系に対する計算モデルにおいて仮定された，システム各部に対する一様・絶対的な観測系が使えないこと意味し，物理学における絶対時空間から相対時空間の使用を，

---

*) 分散型処理系のより簡単な定義として，"複数プロセッサシステムで，相互のコミュニケーションのために共有する中心的記憶場所を用いないもの"[35] などもある。

余裕なく生れた経緯に類似する。 それ故，分散型処理系を整合的に記述のためには，この いわば"相対論的な観点"が，モデルの中に組み込まれるべきである。以下．本節の残りの部分で，ACTORモデルをこの点から考察してみる。

4.2. イベントの局所性

3.2で見たように，ACTOR計算モデルで最も基本的であるイベントは，メッセイジのactorへの到着として定義された。 イベントの定義として，メッセイジの到着でなく，メッセイジの発信を用いなかったこと，又は到着とともに発信もイベントとしなかった理由は，互に物理的交渉のない異った2つのactorからのメッセイジの発信の時間的前後関係を，システム全体に共通して使える大域的時間軸(global time axis)を用いることなしには決定できないという事実である。また．そのような時間的前後関係の比較の必要性も，きわめて少い。(全国的な計算機網が存在すると仮定して，北海道にあるホストからのメッセイジの発信と九州のホストからの発信の前後関係を論ずるより，双方から発信されたメッセイジが，同一の目的地(例えば東京のホスト)に，どちらに先に到着したか否かの方が，より重要であろう。)

3.4で述べた arrival ordering は，各々のactorに到着するメッセイジの到着順を定めるものであったが，ここで注意したいのは，この順序づけは，個々のactorの間で，全く独立に定義されることである。異なるactorに対する arrival ordering が，個々の異った局所時間軸(local time axis)を定める。 よって，異なる2つのactor A_1, A_2 のもつ局所時間軸上の時刻間の比較が可能になるためには，A_1 及び A_2 を標的にもつ2つのイベントの間に，何らかの形で activation ordering による，順序づけが必要となる。

4.3. 情報の流れの局所性

ACTOR計算モデルにおいて，actor間の情報の伝達は，メッセイジによってのみ行われる。3.1で触れたように，actorはそのacquaintancesに属するactorのみにだけ，直接メッセイジを送ることが出来た。 actor A が他のactor Bを(正確にはその名前を)知っている(knows-about)ためには，Aがその誕生時からBを知っているか，以後，何らかのメッセイジを受けとった結果としてBを知るようになったかの，どちらかである。 また，Aが actor C にメッセイジを送って別のactor Dの名前をCに知らしめるには，AがDを知っていなければならない。

このように，ACTOR計算モデルでは，actor間の情報の流れ(それは，他のactorの存在に関する知識の伝達に集約されるが，)に強い制限がある。 この点をもう少し詳しく説明するために，1つのactorに対して，その局所時間軸の上での各時刻で定まる，そのactorの知り得るactorの集合：acquaintance vector [16] を考える。acquaintance vectorは局所時間軸に沿って変化するが，その変化に対する制約を明らかにするために，我々はイベントの参加者(participant)という概念を用いる。

あるイベントEの参加者は次のうちのどれかである：1) Eの標的であるactor，2) その標的のacquaintanceの要素，3) Eのメッセイジの中に含まれる(その名前が言及される)actor，あるいは 4) Eによって生成されるactor。 1つのactorは，誕生したとき，初期の有限なacquaintance vectorをもつ。その要素は誕生の際のイベントの参加者である。 このacquaintance vectorは，そのactorがメッセイジを受けるたびに，そのメッセイジの中で言及されている任意のactor(その数は有限個)を新たな要素として付加

する。 勿論，actorはそのacquaintance vectorの要素を，いつ何時でも忘れることができる。逆に，その時点でのacquaintance vectorの要素を全て，憶えていてよい。こうして，actorはその局所時間軸の時点において，acquaintance vectorの属さないactorにメッセイジを送ることができず，情報の流れに局所性が保たれる。特に，あるイベントEがイベントE'さえ引き起す場合，E'の標的であるactorも，E'のメッセイジの中で言及されるどのactorもEの参加者のひとりでなければならないことになる。

この情報の流れの局所性は，メッセイジの放送(broadcast)によるプロトコールに基づくシステムを排除する。なぜなら，ACTOR計算モデルでは，新しいactorが動的に生成され得るので，あるactorからみて，ある時点での系の中に存在する全てのactorという概念が，そもそも意味をなさないからである。「しかしながら，この情報の流れの局所性は，1つの分散型処理系におけるプロセッサの結合位相の動的変化をACTORモデルで把えるのに何ら問題がないことを注意しておく。

§5§ 分散型処理系/マルチプロセス系の仕様・検証技法

ひとつの分散型処理系が，系全体として，正しく動作するためには，それを構成する各プロセッサのコントロール・プログラム(OS)が最も重要な因子のひとつである。これらのコントロール・プログラムの良い設計，正しい実現のためには，システム全体に要求される挙動がどのようなものであるかを厳密に記述する仕様が是非とも必要である。また，コントロール・プログラムやそのサポート・プログラムが与られたとき，それが仕様に定められたように正しく実現されているか検証されなければならない。特に，分散型処理系の場合，高度の並列処理と，微妙なタイミング・同期に，システム全体のふるまいが依存するため，設計・実現段階でのデッドロックやstarvationの検出・防止に細心の注意を払わねばならない。それには，分散型処理系に適した仕様・検証技法が不可欠になる。

筆者は，ACTORモデルに基づいた並列プログラムの形式的仕様・検証技法を開発した[31]。この技法において，個々のactorの挙動がlocal stateと呼ばれる数学的に定義されたactorの状態を用いて外部的に(externally：インプリメンティションと独立に)記述できる。この技法を用いて，マルチプロセス系で共有・使用されるモデュールの機能挙動の仕様が得られる。この手法に，ある節で述べた，イベント間の順序づけに関する制約条件の，適当な形式的言語による記述[9][10]を併用して，モデュール，プロセス間の相互作用及び，系全体のふるまいに対する仕様が得られる。

このような仕様技法，またこの仕様に基づく検証技法については，ここで立ち入らないが，[31]において，簡単な郵便局の例を用いて，多数の顧客，係員，集配人を並列に走るプロセスとみなして，それらの相互作用や系(郵便局)全体としての機能の仕様(task specification)を示し，その実現を検証する手法が与えられている。また，[35]において，多数の旅行代理店から同時にアクセスできる航空券予約システムの仕様と検証が示されている。さらに多数の異種の入出力装置をタイムシェアリングのユーザが効率よく使うために開発された，サテライト・システムの仕様・検証にもこの技法が用いられる。

・コミュニケイション・プロセシング・システムという比較的固定した対象分野において，仕様・検証の技法がGood等[1][8]によって研究されている。

§6 ACTORモデルの諸応用

前節まで，為ACTOR計算モデルに基づいた分散型処理系を中心に述べてきた。本節において，actorという概念のもう少し基礎的な研究や他の分野における応用について簡単に触れ，本論文におけるACTOR理論の考察を終えることにする。

6.1 オブジェクト指向型言語のセマンティクス

SIMULA[5]，CLU[28]，SMALL-TALK[21]などの抽象データ型*[23][25]の概念をもったオブジェクト指向型言語の単純で明解なセマンティクスとしてACTOR計算モデルを用いることができる。これらのプログラミング言語は，並列処理機能を備えていないので，きわめておおざっぱな言い方をすれば，逆に，ACTOR計算モデルは，これらの言語のセマンティクスとなる計算モデルを，並列性をも含めるように拡張したものであると見ることもできる。ACTOR計算モデルにおけるメッセイジの伝達のメカニズムを忠実に表現することのできるプログラミング言語PLASMA[15]が設計され，MITのAI-Labでその処理系が動いている。3.1で述べたactorのscriptがPLASMAのプログラムとして表現されるわけである。

6.2. 並列性をもつ抽象データ型の形式的仕様

ひとつのactorが受理する要求(request)は固定しているから，データ構造aのようにふるまうactorが，抽象データ型をモデルすることになる。一般にactorでモデルされるモデュールは，多数のプロセスに同時に共有・使用され得ることを前提にしているので，前節で言及したlocal stateを用いて，このようなモデュールの機能・挙動の仕様は，パラレル・プログラムで用いられる抽象データ型の仕様[32][33]とみなされる。抽象データ型の仕様手法は今までに，代数的方法[36][12][7]，公理的方法[29][26][4]，抽象モデルによる方法[17][24]などが提案されているが，並列性(parallelism)をもつ抽象データ型を取り扱えるのは，local stateによるものが初めてであろう。抽象的プロセスについての仕様手法はRiddle[27]によって研究されている。

6.3 概念的表現

local stateを直感的に解りやすく，かつ厳密に表現するために，概念的表現(conceptual representation)[34]と呼ばれる記法系が開発されている。これを用いて，通常の抽象データ型の形式的仕様も書くけるさらに，この概念的表現を用いることにより，オブジェクトの個別性(identity)が容易に表現されるので，オブジェクト指向型の言語で書かれたプログラムの記号値評価(symbolic evaluation)[34]が可能となった。

6.4. 同期問題，制御構造

並列に走るプロセスが，協力して目的を達成するには，プロセス間の同期(synchronization)が，最も重要である。ACTOR計算モデルにおいては，arrival orderingで，メッセイジの到着に線型な順序がつけられることを仮定したが，この性質をつかってメモリセルを用いて同期をとれることが示された[11][9]。さらに，Hoareのモニタ(monitor)[18]の欠陥を改善した同期用メカニズムであるserializer[2]によって，保護したいオブジェクトを囲い込む(encase)方式が開発されている。

---

*) 抽象データ型とは，単なる値の集合によってではなく，それに適用できるオペレイションによって特徴づけられるデータ型である。

3.2で述べたように，ACTOR計算モデルにおけるメッセイジは，継続(continuation)を含むことができる。この継続の選びかたによって，プログラミング言語におけるgoto，手続・関数呼び出し，再帰呼び出し，繰り返し，コルーティンなどの制御構造が実現できることもわかっている[11][15]。

6.5 ビジネス・オートメイションへの応用

actorのような，手続的なオブジェクトは，ビジネス・オートメイションの分野に活用できる。紙による書式やドキュメントのかわりに，TVディスプレイ上のイメイジとして書式をactorで実現し，そのactorに伴った手続，即ちscriptによって，事務員に対して，書式の記入法の指示や，記入された項目の整合性のチェックをさらせる[35]。この"active"な書式が，各部署にある端末間で順次連送される。この方式によって，書式・ドキュメント・システムの柔軟性と保全性およびその処理能率が飛躍的に向上すると思われる。

6.6 アニメイションへの応用

2次元平面上で，複数の比較的単純な物体（登場人物，例えば，Charlie Brown テニスボール，仔犬 etc）が同時に運動し，互に何らかの作用しおよぼし合うアニメイションフィルムは，actorの概念の恰好な応用分野である。Kahnはactorに基づいたアニメイション用の言語を開発し[19]，それを用いたインテリジェントな，アニメイション作製システムをMITのLOGO-Lab.で，実験的な児童教育用ツールとして使おうとしている[20]。

§7§ 参考・引用文献

[1] A. L. Ambler et al., "GYPSY: A Language for Specification and Implementation of Verifiable Programs", ACM Conf. on Lang. Design for Reliable Software, 1977.
[2] R. Atkinson and C. Hewitt, "Synchronization in Actor Systems" SIGPLAN-SIGACT Symp. on Principles of Programming Languages, Los Angels, Jan. 1977.
[3] H. Baker Jr. and C. Hewitt, "The Incremental Garbage Collection of Processes", ACM SIGART-SIGPLAN Symp., Rochester, N.Y., Aug. 1977.
[4] R. Burstall, and J. Goguen, "Putting Theories together to Make Specifications" Int. Jnt. Conf. on Artificial Intelligence, Boston, 1977.
[5] G. Birtwistle, O. J. Dahl, B. Myhrhang and K. Nygaard, SIMULA Begin Auerbach, Philadelphia, 1973.
[6] P. H. Enslow Jr., "What is a Distributed System?", IEEE Computer, Vol. 11, No. 1, Jan., 1978.
[7] J. Goguen, "Algebraic Specification Techniques" Semantics and Theory of Computation Report 9, Dept. of Computer Science, UCLA, 1977.
[8] D. I. Good, "Constructing Verifiable and Reliable Communications Processing Systems", SIGSE Note Vol. 2 No. 5, Oct., 1977.
[9] I. Greif, "Semantics of Communicating Parallel Processes" (Ph.D Thesis) Technical Report TR-154, Laboratory for Computer Science, MIT, Sept. 1975.
[10] I. Greif, "A Language for Formal Problem Specifications", CACM Vol. 20, No. 12, Dec. 1977.
[11] I. Greif and C. Hewitt, "Actor Semantics of PLANNER-73", ACM SIGPLAN-SIGACT Conf., Palo Alto, Jan. 1975.
[12] J. V. Guttag, "The Specification and Applications to Programming of Abstract Data Types"(Ph.D Thesis), Computer System Research Group Report CSRG-59, Univ. of Toronto, 1975.
[13] C. Hewitt, "PLANNER: A Language for Manipulating and Proving Theorems in a Robot", Int. Jnt. Conf. on Artificial Intelligence, Washington, D. C., 1969.
[14] C. Hewitt et al., "A Universal Modular Actor Formalism for Artificial Intelligence", Int. Jnt. Conf. on Artificial Intelligence, Stanford, 1973.
[15] C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages" Journal of Artificial Intelligence, Vol.8, pp.323-364, 1977.
[16] C. Hewitt and H. Baker Jr., "Laws for Communicating Parallel Processes" IFIP-77, Toronto, 1977.
[17] C. A. R. Hoare, "Proof of Correctness of Data Representation" Acta Informatica, Vol.1, pp.271-281, 1972.

[18] C. A. R. Hoare, "Monitors: An Operating System Structuring Concept" CACM Vol.17, No.10, Oct., 1974.
[19] K. Kahn, "An Actor-Based Computer Animation Language", AI-Working Paper, No. 120, Feb. , 1976.
[20] K. Kahn, "Three Interactions between AI and Education", Machine Intelligence 8 Ellis Horwood Ltd., Chichester, Sussex, 1977.
[21] Learning Rsearch Group, "Personal Dynamic Media" SSL-76-1. Xerox Palo Alto Research Center, April, 1976
[22] G. Le Lann, "Distributed Systems -- Towards A Formal Approach --" IFIP-77, Toronto, 1977.
[23] B. Liskov and S. Zilles, "Programming with Abstract Data Types" ACM SIGPLAN Conf. on Very High Level Languages, SIGPLAN NOTICE, Vol.9, No.4, April, 1974.
[24] B. Liskov and V. Berzins, "An Appraisal of Program Specifications" in P. Wegner (Ed.) Research Directions in Software Technoy, MIT Press, Cambridge, 1978.
[25] J. Morris, "Types Are Not Sets" ACM SIGPLAN Symp. on Principles of Programming Languages, 1973
[26] R. Nakajima, M. Honda , and H. Nakahara, "Describing and Verifying Programs with Abstract Types" IFIP Working Conference, New Brunswick, 1977.
[27] W. Riddle, "Abstract Process Types", CU-CS-121-77, Dept. of Computer Science, Univ. of Colorado, Dec., 1977.
[28] C. Schaffert, A. Snyder, and R. Atkinson, "The CLU Reference Manual" Laboratory for Computer Science MIT, Sept. 1975.
[29] J. Spitzen, and B. Wegbreit, "The Verification and Synthesis of Data Structures." Acta Informatica, Vol.4, pp.127-144, 1975.
[30] H. Sullivan, T. R. Bashkow and D. Klappholz, "A Large Scale, Homogeneous, Fully Distributed Parallel Machine, I, II" IEEE Computer Society and ACM 4-th Symp. on Computer Architecture, March 1977.
[31] A. Yonezawa, "Specification and Verification Techniques for Parallel Programs Based on Message Passing Semantics" (Ph.D Thesis) Technical Report TR-191, Laboratory for Computer Science, MIT, Dec. 1977.
[32] A. Yonezawa, "A Specification Technique for Abstract Data Types with Parallelism", Int. Conf. on Mathematical Studies of Information Processing, Kyoto, Aug. 1978. Also available as Research Report C-17, Dept. of Information Science, Tokyo Institute of Technology, April 1978.
[33] 米澤明憲 "並列プログラムにおける抽象データ型の形式的な仕様について" 昭和53年度 情報処理学会第19回全国大会予稿集.
[34] A. Yonezawa, and C. Hewitt, "Syrbolic Evaluation using Conceptual Representations for Programs with Side-effects " AI-Memo, No.399, Artificial Intelligence Laboratory, MIT, Dec. 1976.
[35] A. Yonezawa, and C. Hewitt, "Modelling Distributed Systems" Int. Jnt. Conf. on Artificial Intelligence, Cambridge, Aug. 1977., also in Machine Intelligence 9, Ellis Horwood Ltd., Chichester, Sussex, 1978.
[36] S. Zilles, "Algebraic Specifications of Data Types", Project MAC Progress Report XI, pp.52-58, MIT, Cambridge, 1974.