AIPL（ABCL^c+）と ABCL/1

— 機能とモデルの観点からの比較 —

ABCL^c+ / AIOS 処理系（`yaskodama/abclcp`）

2026 年 07 月 30 日 07:43 JST 版

概要

AIPL は自ら ABCL^c+ と名乗り、ABCL 系の actor-first 並行オブジェクト言語として位置づけられている。本稿は、その源流である ABCL/1（Yonezawa, Briot, Shibayama, OOPSLA 1986）と現在の AIPL を、**機能**と**計算モデル**の 二つの観点から比較する。

結論を先に述べる。AIPL は ABCL/1 から**三種のメッセージ送信**（past / now / future 型、AIPL では `send` / `now` / `future`）をほぼそのまま継承した。一方で ABCL/1 の三つの中核機構 —- **express mode による割り込み、返信先（reply destination）の一級性、script によるパターン受信**— は、いずれも失われているか大幅に縮小されている。その代わりに AIPL が獲得したのは、ABCL/1 が持たなかった**静的型システム**とその健全性の機械検証である。すなわち両者の差は「機能の増減」ではなく、**並行性の表現力を削り、静的な保証を得た**という交換として理解できる。

この交換の帰結として、AIPL には ABCL/1 が express mode と waiting mode で解いていた問題（返信待ちの最中に別の要求を受ける）に対する手段が無い。本稿は最後に、静的型付けを保ったまま何を取り戻せるかを検討する。

目次

1 はじめに

AIPL（ABCL^c+）は、リポジトリ内の型健全性レポートにおいて「ABCL 系の actor-first 並行オブジェクト言語」と自ら位置づけている。また同レポートは形式化の限界を述べる箇所で「ABCM/ABCL では各 actor が一度に一つのメッセージしか処理しない」と、源流のモデルを明示的に参照している。つまり AIPL は ABCL/1 の子孫であることを設計上の前提としている。

そこで本稿は、ABCL/1 と現在の AIPL を並べ、何が受け継がれ、何が落ち、何が足されたのかを整理する。比較は次の二層で行う。

- **機能**：書ける構文と、処理系が提供する機構。
- **モデル**：オブジェクトの状態遷移、メッセージの到達と処理の規律、返信の扱い、並行性の粒度。

AIPL 側の記述は手元の実装（`yaskodama/abclcp`、branch `make-src-base`）を読んで確認した事実に基づく。ABCL/1 側は OOPSLA 1986 論文および MIT Press の論文集 *ABCL: An Object-Oriented Concurrent System*（1990）に基づく。

注意 1. ABCL/1 の**記法**（`<=` などの演算子）については、本稿では処理系を手元で動かして確認していない。記法の細部は一次文献で確認されたい。一方、モデルに関する記述（三つのモード、二重キュー、返信先、script の形）は複数の文献記述で裏づけられるものに限った。

2　ABCL/1 の計算モデル

2.1　並行オブジェクトと三つのモード

ABCL/1 の実行単位は**並行オブジェクト**である。各オブジェクトは常に次の三つのモードのいずれかにある。

モード	状態
dormant（休止）	初期状態。メッセージを待っている
active（活性）	受理したメッセージに応じた動作を実行中
waiting（待機）	実行中に、特定のパターンを満たすメッセージを待っている

dormant なオブジェクトは、script のパターンと制約を満たすメッセージが届くと active になる。一連の動作を終えると再び dormant に戻る。active なオブジェクトは、明示的に「あるパターンのメッセージを待つ」ことができ、このとき waiting になる。該当するメッセージが届けば active に復帰する。

この **waiting モードが重要**である。ABCL/1 では、動作の途中で「この形のメッセージが来るまで待つ」と書けるので、返信待ちや同期を script の中に自然に埋め込める。

2.2　script ―- パターンと制約による受信

オブジェクトは局所記憶（state）と script を持つ。

```
[object object-name
    (state representation-of-local-memory)
    (script
      (=> message-pattern where constraint ...actions...)
      (=> message-pattern where constraint ...actions...))]
```

script は `=>` 節の並びで、各節は**メッセージパターン**、省略可能な **制約**（*where*）、そして**動作**からなる。

ここが ABCL/1 の受信規律である。オブジェクトは「メソッド名で呼ばれる」のではなく、**届いたメッセージが自分の持つどのパターンに合うか**で振る舞いを決める。しかも `where` により、**状態に依存した**受理条件を書ける。たとえば「バッファが空でないときだけ `get` を受理する」を script に直接書ける。これは受理の可否がオブジェクトの状態の関数であるということであり、同期の記述がそのまま受信条件になっている。

2.3　三種のメッセージ送信

ABCL/1 の計算モデルには三種のメッセージ送信が組み込まれている。

種類	送信側の振る舞い	返信
past type	送って即座に先へ進む	受け取らない
now type	返信が来るまで待つ	式の値になる
future type	送って先へ進む	future 変数に届き、後で取り出す

「past（過去）」という名前は、送信側から見て送信がすでに済んだこととして扱われる、という含意である。この三分法が ABCL/1 の最も広く知られた貢献であり、そのまま AIPL の `send` / `now` / `future` に対応する。

2.4 ordinary mode と express mode —– 二重キュー

ABCL/1 の各オブジェクトは**二つのメッセージキュー**を持つ。

- **ordinary mode**：通常のキュー。順に処理される。
- **express mode**：優先キュー。express メッセージが届くと、オブジェクトは**処理中の** *ordinary* **メッセージの処理を中断して** express メッセージを先に処理する。

これは単なる優先度ではなく**割り込み**である。「長い計算をしているオブジェクトに、外から状態を問い合わせる」「暴走しているオブジェクトを止める」といった操作が言語機構として書ける。逐次処理を原則としながら、その原則を破る口を制御された形で用意したのが ABCL/1 の設計判断である。

割り込まれたくない区間は atomicity の宣言で保護できる。すなわち「どこで割り込まれてよいか」がプログラマの制御下にある。

2.5 返信先の一級性

ABCL/1 では**返信先（reply destination）**が値として扱える。now type / future type で送られたメッセージには返信先が伴い、受け取ったオブジェクトはそれを**別のオブジェクトへ渡せる**。

これにより「要求を受けた者が自分では答えず、第三者に答えさせる」という委譲が書ける。パイプライン、代理応答、再帰的な分割統治の末端から直接呼び出し元へ返す、といった構成が可能になる。返信先が一級であることは、単なる糖衣ではなくモデルの表現力そのものである。

2.6 実装

ABCL/1 は Common Lisp の上に構築された（KCL、Symbolics Lisp 上の実装が公開されていた）。したがって型付けは Lisp の動的型付けであり、**静的型検査は存在しない**。

3 AIPL の計算モデル

3.1 アクター＝クラス＋フィールド＋メソッド

AIPL のアクターはクラスから生成される。

```
class Stock {
  var n = 10;                          // 局所状態（フィールド）

  method take(k) : int {               // 名前で dispatch されるメソッド
    n = n - k;
    if (n < 0) { n = 0; }
    reply(n);
  }
}
```

```
var stock = new Stock();
```

ABCL/1 の（state ...）に対応するのがフィールド、script の => 節に対応するのがメソッドである。ただし**受信はパターンではなく名前による dispatch** であり、where に相当する状態依存の受理条件は書けない。

3.2 三種の送信 —- ABCL/1 からの継承

ABCL/1	AIPL	待つか	式か文か
past type	`send a.m(x);`	待たない	文
now type	`now a.m(x)`	返信まで待つ	式
future type	`future a.m(x)`	待たない	式（await で取り出す）

この対応はほぼ一対一である。AIPL は ABCL/1 の三分法をそのまま受け継いだ。違いは、AIPL では now と future の**結果に静的な型が付く**ことである（第 4 節）。

3.3 単一メールボックスと、割り込みの不在

AIPL のアクターは**単一のキュー**しか持たない（実装上 queue : mmessage Queue.t の一本）。express mode に相当する優先キューや割り込み機構は存在しない。実装を検索しても priority / express に相当する語は一つも出てこない。

さらに実行モデルは「1 アクター 1 スレッド」で、アクターのループは一度に一つのメッセージを取り出して評価し終えるまで次に進まない。そして now と await は条件変数で**ブロックする**。

```
let await_future (f:future_state) : value =
  Mutex.lock f.fmutex;
  while f.fresult = None && f.ferror = None do
    Condition.wait f.fcond f.fmutex     (* ここでスレッドが止まる *)
  done;
  ...
```

したがって now で返信を待っているアクターは、その間**他のメッセージを一切処理できない**。ABCL/1 が express mode と waiting mode で解いていた状況に、AIPL は手段を持たない。

この帰結として、now self.m()（自分への同期送信）は必ず自己デッドロックになる。AIPL ではこれが**文法上書けない**ようになっており（送信先は変数名か remote(...) のみで、self は送信先になれない）、危険がひとつ構文で封じられている。しかし「A が now で B を待ち、B が now で A を待つ」という相互デッドロックは依然として書ける。

3.4 返信 —- 暗黙の reply と sender

AIPL の返信は reply(v) で行う。返信先は**暗黙**であり、いま処理しているメッセージに対応する future が解決される。処理系は実行文脈に現在のメッセージ id を持ち、reply はそれを見る。

ABCL/1 のように返信先を値として取り出し、他のオブジェクトへ渡すことはできない。近いものとして sender があり、send sender.m(args); で送信元へ送り返せるが、これは

- 型が any で、メソッドの存在も引数の型も**検査されない**、
- 「返信」ではなく単なる past type 送信であり、呼び出し元の now を解決しない

という二点で、ABCL/1 の返信先とは別物である。委譲は表現できない。

3.5 select —- パターン受信の縮小版

AIPL には select があり、これが ABCL/1 の script に最も近い。

```
select {
  case job(k) -> { reply("done:" ++ k); }
  timeout 500 -> { print("timed out"); }
}
```

メールボックスを走査して、指定した名前のメッセージを選んで処理する。ABCL/1 の waiting モードに対応する機能である。ただし

- 受理条件はメッセージ**名**のみで、where に相当する状態依存の制約は書けない、
- 通常のメソッド dispatch と競合する。同名のメッセージが通常の dispatch で先に処理されてしまうと、select には残らない、
- timeout を持つ（ABCL/1 の script には無い機構）

という違いがある。二番目は実際に観測できる。send w.serve(); で select を待たせたうえで now w.job(1) を送ると、job が通常の dispatch で先に処理され、select は timeout に落ちる。

```
direct:1
waiting
timed out
```

script が**唯一の**受信経路であった ABCL/1 では、この競合は起こりえない。AIPL は「名前による dispatch」を主とし、select を副として後から足したため、二つの受信経路が並立している。

3.6 become

AIPL には become C(args); があり、アクターの振る舞いを別クラスへ置き換えられる。これは Hewitt 系 actor の become に近い機構で、ABCL/1 が状態変数と script の制約で表現していたものを、別の形で提供している。

ただし現状の型検査は become のコンストラクタ引数しか見ておらず、置換後のインタフェースが元と一致するかを検査しない。

3.7 境界 —- 分散と Web

AIPL は ABCL/1 に無かった機構を二つ持つ。

- remote("host:port", "actor") を送信先に取れる。ノードをまたいだ送信が言語に組み込まれている。
- web_listen(port) で HTTP ゲートウェイが起動し、POST /api/send で外部から任意のアクターへメッセージを送れる。web_expose は公開エンドポイントに別名を付ける。

ABCL/1 は ABCL MIMD システム、すなわち**並列計算機**のための言語であり、広域分散や外部プロトコルは想定範囲外だった。AIPL のこの二つは、時代の要求に応じた追加である。同時に、これが AIPL の型システムにおける最大の `any` 境界にもなっている。

3.8 実装

AIPL は OCaml で実装されている。字句解析器・構文解析器・型検査器・評価器・HTTP ゲートウェイが自前で、Lisp の上に載っていない。これは静的型検査を持つうえで自然な選択である。

4 型 —– 最大の違い

4.1 ABCL/1 は動的、AIPL は静的

ABCL/1 は Common Lisp の上に構築された動的型付け言語であり、静的型検査を持たない。メッセージが受理されるかは実行時に script のパターンと制約で決まる。

AIPL は静的型システムを持つ。今回の改修により、特にメッセージ通信の部分に型が付くようになった。

検査	内容
メソッドの存在	ローカル送信先にその名前のメソッドがあるか
引数の個数と型	実引数と**本体が要求する型**が一致するか
返信の型	すべての `reply` が同じ（宣言された）型か
返信の回数	高々一度か。注釈があれば全パスでちょうど一度か
`select`	`case` の arity と引数型が署名と一致するか
公開時の注釈	外部公開したアクターのメソッドに戻り値注釈があるか

とくに**返信の型と回数**は、ABCL/1 のモデルには対応する概念が無い。ABCL/1 の返信先は実行時の値であり、そこへ何を何回送るかは動的な問題である。AIPL はメソッドごとに戻り値型 $\rho_{C,m}$ を一つ持ち、本体のすべての `reply` と、すべての `now` / `future` 送信地点がこの型を共有する。

4.2 三種の送信に型を付けるということ

`send` $a.m(\vec{e})$	: `unit`（文）
`future` $a.m(\vec{e})$	: `future` $\rho_{C,m}$
`now` $a.m(\vec{e})$	: $\rho_{C,m}$
`await` e	: τ $\quad (e : \texttt{future}\ \tau)$

ABCL/1 の三分法は**制御の**分類だった。AIPL ではこれが**型の**分類にもなっている。past type は返信を持たないので戻り値型が `unit` に落ち、now / future は同じ ρ を共有し、future だけが `future` で包まれる。

4.3 ABCL/f —- 同じ問題への先行例

型と future を組み合わせる試みは ABCL 系に先例がある。Taura, Matsuoka, Yonezawa の **ABCL/f**（1994）は “A Future-Based Polymorphic Typed Concurrent Object-Oriented Language” と題されており、future を中心に据えた多相型付き並行オブジェクト言語である。

すなわち「ABCL に型を付ける」という方向自体は 1990 年代に既に取られていた。AIPL の今回の作業は、その系譜の中では新規というより**実装済みの動的な処理系に、後から型を入れていく**という別の入り方である。その代わり AIPL は、型健全性（保存・進行）を Coq で機械検証し、さらに**型検査器が付けた型と実行時の値を突き合わせる差分テスト**を常設した。後者は「型システムと実装のずれ」を自動で見つける仕組みで、実際に整数演算が `float` を返していた不整合をこれで発見している。

注意 2. AIPL の多相性は限定的である。メソッドの引数型変数は一般化されないため、同じメソッドを `int` と `string` で呼ぶと衝突する。ABCL/f が掲げた polymorphic には届いていない。これは戻り値型 ρ を型スキームに埋めると呼び出しごとに差し替えられてしまう、という実装上の制約に由来する。

5 機能比較

	ABCL/1	AIPL（ABCL^c+）
実行単位	並行オブジェクト	アクター（クラスから生成）
オブジェクトのモード	dormant / active / waiting の三態	明示的なモード概念は無い。1 アクター 1 スレッドで逐次処理
受信の規律	script のパターン＋ `where` 制約	メソッド名による dispatch。副として `select`
状態依存の受理	`where` で書ける	書けない
past type	あり	`send`
now type	あり	`now`（式）
future type	あり	`future` ＋ `await`
express mode	**あり**（割り込み、二重キュー）	**無し**
atomicity 宣言	あり（割り込み禁止区間）	無し（そもそも割り込みが無い）
返信先	**一級の値**。委譲できる	暗黙。`sender` は `any` で未検査
timeout	script には無い	`select` の `timeout`
振る舞いの置換	状態変数と制約で表現	`become`（インタフェース未検査）
分散・外部境界	並列計算機が対象	`remote(...)`、HTTP ゲートウェイ
反射	ABCL/R、ABCL/R2 として別系統	`aios_*` による内省プリミティブ
型付け	動的（Common Lisp 上）	**静的**。HM 風推論＋戻り値型注釈、健全性を Coq で検証
実装言語	Common Lisp（KCL、Symbolics）	OCaml（自前の全段）

6 モデルの観点からの本質的な差

機能表の裏にある、モデルとしての差は四点に整理できる。

6.1 割り込み可能性 —- 逐次性の破り方

ABCL/1 も AIPL も「一つのオブジェクトは一度に一つのメッセージ」を原則とする。違いは**その原則を破る手段があるか**である。

ABCL/1 は express mode という制御された抜け道を用意し、どこで割り込まれてよいかを atomicity 宣言でプログラマに委ねた。AIPL には抜け道が無い。`now` でブロックしたアクターは石になる。

これは「型安全性には影響しない」（型は状態が正しい型であることしか言わない）が、**進行性**に直接影響する。実際 AIPL の健全性レポートは進行定理を「値・簡約・await 待ち」の三択として述べており、await 待ちで止まることを定理の中で認めている。デッドロックは型だけでは排除できないという境界の明示であり、ABCL/1 はそこを言語機構（express mode）で埋めていた。

6.2 返信先の一級性 —- 委譲できるか

ABCL/1 の返信先は値である。したがって「A が B に投げ、B が答えずに C に返信先を渡し、C が A に返す」と書ける。これは継続を渡すのに等しく、パイプラインや分割統治を自然に表現できる。

AIPL の返信先は暗黙で、取り出せない。同じ形を書くには「B が C を `now` で待って中継する」しかなく、その間 B は塞がる。すなわち**委譲が中継に退化する**。これは表現力の差であると同時に、上の割り込み問題と結びついて性能・進行性の差にもなる。

6.3 受信の選択 —- 同期を受信条件として書けるか

ABCL/1 の `where` は、受理の可否をオブジェクトの状態の関数にする。「バッファが空でないときだけ `get` を受理する」がそのまま書ける。同期の記述と受信条件が一致しているので、待ち合わせのためのフラグやループを書かなくてよい。

AIPL は名前で dispatch するので、状態に依存した受理ができない。`select` で名前は選べるが条件は付けられず、しかも通常の dispatch と競合する。結果として、状態依存の待ち合わせは「受理してから条件を見て、駄目なら自分に送り直す」という形になり、受信規律の外へ出てしまう。

6.4 型 —- 保証を得た代償

三点はいずれも AIPL が**失った**ものである。得たものが型である。

ABCL/1 では「このメッセージを送ったら何が返るか」は実行時の問題だった。AIPL では、メソッドごとの戻り値型が本体の `reply` と送信地点を結び、`now` の結果が具体的な型を持つ。さらに

- 返信は高々一度（注釈があればちょうど一度）、
- 外部公開するメソッドは戻り値型の宣言が必須、
- 引数の型は本体が要求する型と照合される

といった規律が静的に強制される。ABCL/1 のモデルには、これらを述べる語彙そのものが無い。

注意 3. 興味深いのは、**失った表現力が型付けを容易にしている**ことである。express mode による割り込みがあると、メソッド本体の途中で状態が書き換わりうるので、状態の型不変条件を保つ議論が難しくなる。返信先が一級だと、返信の型は「誰が最終的に返すか」に依存し、メソッドごとの ρ という単

純な形に収まらない。状態依存の受理条件は、型ではなく依存型・篩型の領域である。AIPL の型システムが 200 行程度で成立し、Coq で検証できたのは、モデルを削ったからでもある。

7　何を取り戻せるか

削った表現力のうち、静的型付けを保ったまま取り戻せそうなものを検討する。

機構	型付けとの折り合い
返信先の一級化	返信先を `Reply`$\langle\tau\rangle$ という型の値にする。Akka Typed の `ActorRef[R]` と同じ形で、型は付く。**ただし**「ちょうど一度返す」を保つには線形型が必要になる。現在の構文的な回数検査は、返信先が値になった途端に効かなくなる
状態依存の受理	`where` を型で扱うには依存型・篩型が要る。現実的には「受理条件は動的に検査する」と割り切り、型は**メッセージの形**だけを保証する分業が妥当
express mode	型安全性への影響は小さい（状態の型は保たれる）。むしろ問題は**状態不変条件**であり、割り込み可能点を宣言させる（ABCL/1 の atomicity）方向なら型システムを壊さずに入れられる
`select` の統一	現状の「名前 dispatch と `select` の並立」は競合を生んでいる。script のように受信経路を一本化すれば、型検査もメッセージ形の照合として一元化できる
多相メソッド	ABCL/f が掲げた方向。戻り値型 ρ を型スキームに含めると呼び出しごとに差し替えられてしまうので、ρ と引数型を別扱いにする定式化が必要

優先順を付けるなら、**受信経路の一本化**が最も費用対効果が高い。これは表現力を増やす話ではなく、AIPL が後から `select` を足したことで生じた不整合を解消する話であり、ABCL/1 の script に近づく方向でもある。

次に**返信先の一級化**である。これは ABCL/1 との差のうち最も本質的なもので、しかも Akka Typed の前例があるため型付けの見通しが立つ。線形性の扱いが焦点になる。

express mode は、型よりも状態不変条件の問題として扱うのが筋である。

8　まとめ

AIPL は ABCL/1 から三種のメッセージ送信を継承し、express mode、返信先の一級性、script によるパターン受信を落とし、静的型システムと分散・Web 境界を足した言語である。

この交換は偶然ではない。落とした三つはいずれも**メッセージの処理順序や返信の行き先を動的にする**機構であり、静的な型付けと素直には両立しない。AIPL が型健全性を機械検証できる規模に収まったのは、モデルを単純化したからでもある。

したがって「ABCL/1 の方が表現力が高く、AIPL の方が安全である」というのは正しいが、それだけでは片面的である。より正確には、**ABCL/1 は並行性の記述に賭け、AIPL は静的な保証に賭けた**のであり、両者は同じ三分法を共有しながら異なる方向へ最適化した親子である。

残された課題は、この二つを対立ではなく段階として扱えるか—- すなわち ABCL/1 が持っていた機構を、型が保証できる形で言語に戻せるか —- である。本稿の第 7 節がその見通しである。

参考文献

[1] Akinori Yonezawa, Jean-Pierre Briot, Etsuya Shibayama. Object-Oriented Concurrent Programming in ABCL/1. *OOPSLA 1986*, pp. 258–268. `https://dl.acm.org/doi/10.1145/28697.28722`

[2] Akinori Yonezawa (ed.). *ABCL: An Object-Oriented Concurrent System*. MIT Press, 1990.

[3] Kenjiro Taura, Satoshi Matsuoka, Akinori Yonezawa. ABCL/f: A Future-Based Polymorphic Typed Concurrent Object-Oriented Language. 1994.

[4] Takuo Watanabe, Akinori Yonezawa. Reflection in an Object-Oriented Concurrent Language (ABCL/R). *OOPSLA 1988*.

[5] Naoki Kobayashi, Akinori Yonezawa. Type-theoretic foundations for concurrent object-oriented programming. *OOPSLA 1994*.

[6] Actor-Based Concurrent Language. `https://en.wikipedia.org/wiki/Actor-Based_Concurrent_Language`

[7] ABCL/1 — FOLDOC. `https://foldoc.org/ABCL/1`

[8] 本リポジトリ `docs/aipl_soundness_ja.pdf` — AIPL^{-} の型健全性・型安全性の Coq 検証。

[9] 本リポジトリ `docs/aipl_reply_inference_ja.pdf` — `reply` からの戻り値型推論、戻り値型注釈、リモート境界での注釈必須化、測定結果。

[10] 本リポジトリ `docs/aipl_guide_ja.pdf` — 現状の言語ガイド。