AICE のメタ環境に関する研究

Actor 型、Protocol、Tool、Memory を統合する自己記述的 AIOS

Yasushi Kodama Repository Report

2026 年 5 月 4 日

概要

本稿は、AICE（Actor-based Intelligent Computing Environment）を単なる AI application 実行基盤としてではなく、自己の構成要素を観測・記述・検査・再構成できるメタ環境として捉える研究構想を述べる。AICE は AIPL によって記述される型付き actor 環境であり、AI agent、model adapter、tool adapter、memory service、workflow protocol、UI、OS service を actor として統一的に扱う。メタ環境とは、これらの actor 群を対象として、構成、型、capability、protocol、実行 trace、性能、失敗、変更履歴を第一級データとして扱う上位環境である。

本稿では、AICE メタ環境を、reflection、introspection、orchestration、verification、evolution の五つの機能を持つ layer として設計する。特に、AIPL の型情報、actor registry、session protocol、event log、memory store を統合することで、AI system が自己の workflow を監査し、失敗を診断し、actor 構成を再編成する仕組みを提案する。

目次

1　はじめに

AICE は、AI agent、model、memory、tool、UI、OS service を actor として構成する知的計算環境である。従来の AI application では、これらの構成要素は program 内の関数、外部 API call、設定ファイル、prompt template として散在しやすい。その結果、system がどの actor から構成され、どの actor がどの capability を持ち、どの workflow protocol に従っているかを把握することが難しくなる。

AICE のメタ環境は、この問題に対して、AICE 自身を観測・記述・検査・変更可能な対象として扱う。つまり、AI system を実行する環境だけでなく、AI system の構造を扱う環境を作る。

2　メタ環境の定義

定義 1 (AICE メタ環境)**.** *AICE* メタ環境とは、*AICE* 内の *actor*、型、*capability*、*protocol*、*event*、*memory*、*tool*、*model*、*configuration* を第一級データとして表現し、それらを観測、検査、変更するための *actor* 群である。

メタ環境は、object-level AICE と meta-level AICE の二層からなる。

$$MetaAICE \curvearrowright ObjectAICE$$

ObjectAICE は通常の AI workflow を実行する。MetaAICE は ObjectAICE の registry、type environment、protocol state、event log、resource usage を監視し、必要に応じて診断や再構成を行う。

3　メタ環境が必要な理由

AIOS が大規模化すると、単に actor を実行できるだけでは不十分である。必要になるのは、system 自体を理解し、変更し、検査する能力である。

3.1　構成の可視化

複数の model actor、memory actor、tool actor、agent actor があるとき、どの actor がどの actor に依存しているかを可視化する必要がある。

$$ActorGraph = (Actor, MessageEdge, CapabilityEdge)$$

3.2　失敗診断

AI workflow は失敗する。model が不正確な応答を返す、tool call が失敗する、protocol 順序が誤る、memory が古い情報を返す、などである。メタ環境は event log と protocol state を使って、失敗箇所を actor graph 上で特定する。

3.3 自己再構成

ある model actor が失敗し続ける場合、router actor は別の model actor へ切り替えるべきである。ある reviewer が弱い場合、追加 reviewer を workflow に挿入するべきである。これらは object-level の処理ではなく、meta-level の orchestration である。

4 メタ環境の構成 actor

表 1: AICE メタ環境の actor

Actor	役割
RegistryInspector	actor registry と method table を読む
TypeInspector	AIPL 型環境、method signature、future 型を検査する
ProtocolInspector	session protocol の定義、現在位置、違反履歴を読む
TraceAnalyzer	event log、message trace、tool log を解析する
CapabilityAuditor	actor が持つ capability と実行済み primitive を照合する
WorkflowSynthesizer	goal から actor workflow 候補を生成する
FailureDiagnoser	失敗した actor、message、protocol step を推定する
ReconfigurationActor	actor の差し替え、追加、停止、再起動を行う
MetaMemory	system 構成、失敗履歴、改善履歴を保存する

5 自己記述的 actor

メタ環境を実現するためには、actor が自分の interface を記述できる必要がある。AIPL actor は次のメタ情報を公開する。

- actor name
- class name
- method signature
- capability requirements
- current mailbox length
- protocol participation
- resource usage

たとえば、model actor は次のような記述を持つ。

```
ModelActor {
  generate  : Prompt -> future Response
```

```
  capability: Model.Call
  provider  : OpenAI | Gemini | Local
}
```

この自己記述性により、meta actor は workflow を型と capability に基づいて合成できる。

6 型情報のメタ利用

AICE メタ環境では、型は実行前検査だけでなく、system 理解のための metadata になる。

$$TypeInfo(actor.method) = \tau_1 * \cdots * \tau_n \to \tau_r$$

WorkflowSynthesizer は、出力型と入力型を接続して actor chain を生成できる。

$$Problem \xrightarrow{planner.plan} Plan \xrightarrow{solver.solve} Candidate \xrightarrow{reviewer.review} Review$$

型がなければ、この合成は名前や自然言語説明に依存する。型があれば、少なくとも接続可能性を機械的に判定できる。

7 protocol のメタ管理

ProtocolInspector は、session protocol を object-level workflow の trace と照合する。

$$ExpectedTrace(P) = [a_1.m_1, \ldots, a_n.m_n]$$

$$ActualTrace = [b_1.k_1, \ldots, b_m.k_m]$$

ProtocolInspector は、ActualTrace が ExpectedTrace の prefix かどうかを検査する。違反があれば、FailureDiagnoser が次を特定する。

- どの actor が予期しない message を送ったか。
- どの protocol step が未消費か。
- どの actor が応答しなかったか。
- 再試行すべきか、再計画すべきか。

8 メタ環境による自己改善

AICE メタ環境は、失敗から system を改善する。

$$FailureTrace \to Diagnosis \to ReconfigurationPlan \to UpdatedWorkflow$$

例として、solver が不完全な解を返し reviewer が拒否した場合、メタ環境は次の改善を行える。

1. solver prompt を変更する。

2. 別の model actor へ切り替える。
3. tool actor を workflow に追加する。
4. memory retrieval を solver の前に挿入する。
5. reviewer を二重化する。

この改善は、単なる retry ではなく、trace と type と protocol に基づく再構成である。

9 安全性

メタ環境は強力であるため、安全性が必要である。ReconfigurationActor が任意の actor を停止・置換できるなら、system 全体が危険になる。したがって、メタ操作にも capability と protocol を導入する。

$$MetaCapability ::= Inspect \mid Diagnose \mid Restart \mid Replace \mid RewriteProtocol$$

たとえば、通常の TraceAnalyzer は Inspect capability のみを持つ。ReconfigurationActor は Replace capability を持つが、その操作は監査 log に記録される。

10 実装計画

表 2: AICE メタ環境の実装段階

段階	内容
Stage 1	actor registry、method table、mailbox length の introspection
Stage 2	event log と protocol state の統合表示
Stage 3	type environment を用いた workflow 接続検査
Stage 4	failure diagnosis actor の導入
Stage 5	workflow synthesizer による actor chain 生成
Stage 6	capability 付き reconfiguration actor の導入

11 結論

AICE のメタ環境は、AIOS を単なる実行基盤から、自己を観測し、検査し、改善できる知的計算環境へ拡張する。AIPL の型、actor registry、capability、protocol、event log、memory を統合することで、AI workflow の構造と実行過程を第一級データとして扱える。

今後の課題は、メタ操作の安全性、workflow 自動合成の正当性、失敗診断の精度、そして自己再構成が system の安定性を損なわないための protocol 設計である。