DOFBOT × Xinu メッシュ 6 台

システム全体設計（第 2 版）

Capability 推論による役割分担・冗長化・アクター移動

2026 年 7 月 21 日

Abstract

第 1 版は 12 台を静的な ServoMotor/ServoSensor 対として扱い、当初構想の中核である**Capability 推論による役割分担・冗長性・アクターの動的移動**を設計に反映していなかった。本版はそれを中心に据えて構成し直す。

中心となる主張は 1 つ。**アクターの効果集合から、配置・複製可否・フェイルオーバの手順が機械的に決まる**。純粋なアクターは自由に複製でき、外部作用を持つアクターは**同時に 1 つしか存在してはならない** — つまりフェイルオーバとは Capability の移譲そのものである。

6 軸に 6 台を 1 対 1 で割り当て、**冗長性は待機機ではなく、故障時に空いているノードへアクターを転送することで確保する**。6 台で全軸を担当でき、当初構想の中核である**アクターの動的移動がそのまま可用性の機構になる**。

ただし現状 2 つの障害がある。**効果格子が「自己状態の変更」と「外部への作用」を同一視している**こと、そして**実測レイテンシの裾が秒単位**であること。前者は型システムの小さな拡張で解ける。後者は解けないので、**フェイルオーバは秒オーダである**という前提で安全性を設計する。

1　当初構想と第 1 版のずれ

当初の構想は次のとおりであった。

> AIPL のアクターを生成し、Capability 推論で役割分担させ、6 軸に対して 2 台ずつ Xinu を用意して冗長性を持たせ、アクターの移動を動的に行い、全体として効率の良いリアルタイム OS とする。

第 1 版はこのうち「6 軸 × 2 台」だけを写し取り、残りを落としていた。Capability は役割の**説明**に使われるだけで設計判断に効いておらず、冗長性は台数の話に留まり、アクター移動に至っては触れていない。本版で立て直す。

2　現物 — CG シミュレータ

図 1 が本設計の出発点である。重要なのは右下のパネルで、**効果はソースに書かれたものではなく推論結果**であり、実処理系の `--type-check` が申告と実体の一致を検査する。本設計はこのパネルが示す情報だけから配置・複製・フェイルオーバを導く。

シミュレータは 6 軸それぞれに soft-Xinu を 2 台、計 12 個のアクターを走らせている。**実機 6 台は、このうち 6 個を置き換える**という位置づけになる。

3　6 台の割当 — 1 軸 1 台

6 軸に 6 台を 1 対 1 で割り当てる。アーム全軸が実機 Xinu で駆動される。

Figure 1: DOFBOT ライン作業シミュレータ（`:8022`）。2026-07-21 実行、`node test/verify.mjs` で全項目合格した状態。左は 3D の作業セル（銀色ホーンの丸数字が実機サーボ ID 1–6 に対応）、右上は手首カメラの映像と TinyML の分類結果（緑・六角柱 100 %）、中央右は**実行中の AIPL ソースで、現在実行している行が強調表示**される（`PlannerActor` 268 行目）、右下が **Actor / Capability パネル**で、各アクターの効果推論結果 `!{mut} !{net}` がチップとして表示される。

ノード	担当	役割	保持する効果
1	ID1 ベース旋回	`ServoMotor`	`!{act, mut, net}`
2	ID2 肩	`ServoMotor`	`!{act, mut, net}`
3	ID3 肘	`ServoMotor`	`!{act, mut, net}`
4	ID4 手首ピッチ	`ServoMotor`	`!{act, mut, net}`
5	ID5 手首回転	`ServoMotor`	`!{act, mut, net}`
6	ID6 グリッパ	`ServoMotor`	`!{act, mut, net}`

対応する `ServoSensor` 6 個は CG 側の soft-Xinu が担う。純粋アクター（`Kinematics`）は効果を持たないので **6 台すべてに複製**する（第 4.3 節）。

冗長性は待機機を置くのではなく、いざというときに空いているノードへアクターを転送して確保する。 6 台すべてが実務に就いているので、専用の待機機は存在しない。故障時には、**生き残ったノードのうち余裕のあるものが失われたアクターを引き受ける**。次節で成立条件を示す。

台数効果の測定点は **2 / 4 / 6** の 3 点とする。担当する軸数を増やしながら同じ作業のサイクル時間を測れば傾向が取れる。1 点では傾向を主張できないので、**6 台は「3 点が取れる最小」**でもある。

4 Capability から配置と冗長化を導く

4.1 効果は宣言ではなく推論される

AIPL の Capability はソースに書くものではない。各メソッド本体の一次作用を効果推論が走査して `!{mut, net, ai, fs}` を導出し、`--type-check` が申告と実体の食い違いを検出する。現在の割当は次のとおり。

一次作用	効果	意味
`Arm_serial_servo_write`	`mut`	実機バスサーボの駆動
`Arm_serial_servo_read`	なし	計測のみ。何も動かさない
`tinyml_infer`	`ai`	ローカル推論。`net` を伴わない = 機外へ出ない
`tinyml_load` / `write_file`	`fs`	記憶装置
`send` / `now` / `reply`	`net`	メッセージ

これは強力である。`ServoSensor` は `Arm_serial_servo_write` を呼ぶ場所がソース中のどこにも無いため、**型によって物理的に駆動できないことが保証される**。最小権限が設計意図ではなく検査可能な性質になっている。

4.2 発見: 効果格子が作用と状態変更を同一視している

しかし配置の自動導出には足りない。現在のクラスの効果は次のようになる。

クラス	効果	外部への作用
`ServoMotor`	`!{mut, net}`	**あり（サーボを回す）**
`ShelfActor`	`!{mut, net}`	なし（自分の在庫表を書くだけ）
`ConveyorActor`	`!{mut, net}`	あり（ストッパを開閉）
`ArmMover`	`!{net}`	なし
`Kinematics`	`!{net}`	なし（純計算 + `reply`）
`RecognitionActor`	`!{ai, net}`	なし
`SafetyActor`	`!{fs, mut, net}`	なし

`ServoMotor` と `ShelfActor` は効果集合が同一である。 一方は実機を動かし、他方は自分のフィールドを書くだけなのに区別が付かない。ソース中のコメント自身が「`bind()` の `mut` は自分のフィールド `id` への代入であって、サーボ駆動ではない」と断っており、この曖昧さは認識されている。

配置と複製の規則を効果から導くには、この 2 つを分ける必要がある。

提案する効果	意味	導かれる規則
`mut`	自分の状態を書く	複製可（各複製が自分の状態を持つ）
`act`	**外部世界へ作用する**	**同時に 1 つだけ**。配置はその機器に繋がったノードに限定

`Arm_serial_servo_write` と `conveyor_stopper` を `act` に移すだけで、`ServoMotor`/`ConveyorActor` と `ShelfActor` が型で分かれる。**これは小さな変更だが、以降の設計全体がこの区別の上に載る。**

4.3 効果集合から決まる規則

効果	配置	複製	根拠
なし（純粋）	どこでも	**自由**	副作用が無いので二重実行が無害。冗長化の費用がゼロ
`!{net}`	どこでも	可（冪等なら）	重複メッセージを受け手が吸収できる必要
`!{mut}`	どこでも	可	各複製が独立した状態を持つ
`!{ai}`	重みを持つノード	可	推論は決定的。機外へ出ない
`!{fs}`	記憶装置を持つノード	不可	書き込みが競合する
`!{act}`	**その機器に繋がったノード**	**不可（厳密に 1 つ）**	二重駆動は物理的に危険

`Kinematics` は効果を持たないので**全ノードに複製してよい**。逆運動学を 6 台のどこで解いても同じ答えが出るので、冗長化は無料である。一方 `ServoMotor` は `act` を持つので**厳密に 1 つ**でなければならない。

5 冗長化 — 空きノードへのアクター転送

専用の待機機を置かず、**故障時に失われたアクターを別のノードへ転送する**ことで冗長性を確保する。6 台すべてが実務に就いたまま冗長化できるのが利点である。

5.1 成立条件 1 — `act` が場所に縛られないこと

第 4.3 節の規則では、`act` を持つアクターの配置は「その機器に繋がったノード」に限定される。もしこの制約が効くなら、`ServoMotor` は特定のノードから動かせず、転送による冗長化は**成立しない**。

本構成では制約が効かない。 サーボバスはアーム内蔵 Pi（L0）の背後にあり、Xinu 側はネットワーク越しに駆動指令を送る。**どのノードからでも同じように届くので、`act` の配置は場所に依存しない。**

これは本構成に固有の性質であって、一般的な性質ではない。 別紙で提案した「スマートジョイント」構成 — 各関節のボードが自分のサーボに直接配線される — では `act` は物理的に場所に縛られ、**この転送方式は使えなくなる**。冗長化は待機機方式へ戻る必要がある。設計を実機構成ごと切り替えるときに、ここを見落としてはならない。

5.2 成立条件 2 — 状態が機器から読み直せること

転送先のノードは失われたアクターの状態を持っていない。待機機方式なら副が鏡像を保持しているので転送が要らないが、今回は**状態をどこかから復元しなければならない**。今日の実測ではメッシュの裾が秒単位あるため、故障したノードから取りに行く設計はそもそも成立しない（相手は落ちている）。

`ServoMotor` の状態は機器から読み直せる。 このアクターが持つのは `id` と `pos` だけで、`pos` は `Arm_serial_servo_read` で現在角として取得できる。つまり**状態はハードウェアの側にある実体の写しにすぎず、転送先で再取得できる。**

アクター	状態の復元	転送可否
`ServoMotor`	**機器から読み直す**	**可**（状態転送が不要）
`Kinematics`	状態を持たない	**可**（そもそも複製済み）
`ShelfActor`	**機器に無い**	要状態転送。棚の在庫は外から読めない
`SafetyActor`	`fs` に残る	記憶装置を持つノードに限定

したがって転送方式が無条件に使えるのは `ServoMotor` と純粋アクターであり、それ以外は別途手当てが要る。 6 軸の駆動という本題に関しては十分である。

5.3 成立条件 3 — 生き残りに余裕があること

6 台すべてが 1 軸を担当しているので、**空いているノードは存在しない**。故障時は**生き残ったノードのいずれかが 2 軸を兼務する**ことになる。

状態	ノード数	担当
正常	6	各ノード 1 軸
1 台故障	5	1 台が 2 軸、残り 4 台が 1 軸
2 台故障	4	2 台が 2 軸、残り 2 台が 1 軸

これは「各ノードが 2 軸ぶんの実時間予算を持つこと」を要求する。 成立するかは実測でしか分からない。10 Hz 制御なら 1 軸あたりの処理は推論を含めても 1 ms 未満（第 7 節）なので余裕はあるはずだが、**兼務時の締切遵守率を測るまでは仮定である**。これは 6 台構成で最初に測るべき数字の 1 つである。

5.4 転送の手順 — `act` は期限付きリース

`act` は厳密に 1 つという規則があるので、転送は**排他的な移譲**でなければならない。分断時に新旧 2 つの `ServoMotor` が同じ軸を駆動する事態は物理的に危険である。そこで `act` を**期限付きリース**として扱う。

1. 保持者は周期 T_r でリースを更新する。
2. 保持者が T_e 以内に更新できなければ、**自力で駆動を停止する**（ネットワーク不要。ローカルの時計だけで判定できる）。
3. 協調役が T_e 経過を観測し、余裕のあるノードを選んで `/actor/load` でコードを送り込む。
4. 転送先が `Arm_serial_servo_read` で現在角を読み、状態を復元する。
5. 転送先が `act` リースを取得し、駆動を開始する。

手順 2 が安全性の要である。 分断された旧保持者が「自分はまだ担当だ」と思い込んで駆動し続けることを、ネットワークに頼らず防ぐ。分散ロックのリースと同じ論法で、**Capability がリソースではなく期限付き権限であることを利用している。**

5.5 タイミング — 秒オーダであることを認める

量	値	根拠
HELLO 周期	2 s	`device/wifi/wifi.c`（現行実装）
往復レイテンシ p99	84–1019 ms	本日実測（Pi 5 / Pi 3、Pi 3 は `NTCP` 修正後）
リース更新 T_r	≥1 s	p99 の裾を吸収する
リース期限 T_e	≥3 s	T_r の 3 倍。誤検知を避ける
コード転送 `/actor/load`	**38–115 ms**	実測 (2026-07-21)。Pi 5 中央値 37.9 ms / Pi 4 114.8 ms。1743 B の Se
状態の読み直し	~1 往復	`Arm_serial_servo_read`
フェイルオーバ所要	**3–5 s**	上記の合計。**コード転送は実測で 0.04–0.12 s にすぎず**、支配するの

したがってこの冗長化は「可用性」のためであって「無停止」のためではない。 3–5 秒のあいだ、その軸には駆動権限を持つ者がいない。

当初 4–8 秒と見積もったが、実測でコード転送が安いことが分かり 3–5 秒に下がった。 `/actor/load` の所要時間は Pi 5 で中央値 37.9 ms、Pi 4 で 114.8 ms （1743 B の `ServoMotor` を投入し常駐を確認）。しかもその大半は JIT ではなく HTTP 自体の遅延で、Pi 5 の通常 HTTP が 38 ms、Pi 4 が 130 ms であることと符合する。**コード移動の費用は無視できる** — 転送方式の弱点は転送そのものではなく、故障検知に要するリース期限である。**その間アームが安全であることは、メッシュではなく L0（アーム内蔵 Pi）が保証しなければならない** — 最後の目標角を保持するか、停止する。これは選択肢ではなく要件である。

5.6 待機機方式との比較

	転送方式（本設計）	待機機方式
6 台での担当軸数	**6 軸すべて**	3 軸のみ
状態の復元	機器から読み直す	副が鏡像を保持（不要）
フェイルオーバ	3–5 s（実測反映）	3–5 s
故障耐性	台数ぶん（縮退運転）	各軸 1 台まで
前提	**`act` が場所に縛られないこと**	なし
実証できる主張	**アクターの動的移動**	昇格のみ

転送方式を採る。 6 台で全軸を担当でき、しかも当初構想の中核である **アクターの動的移動そのものが冗長化の機構になる**ためである。フェイルオーバが数秒遅い点は、L0 側の安全保持で吸収する。

6 アクターの動的移動

6.1 既にあるもの・無いもの

要素	状態	備考
コード移動	**実装済**	`/actor/load`（AIPL → C JIT）が Pi 4 / Pi 5 にある
効果推論	**実装済**	`--type-check` で申告と実体を照合
`act` の分離	**未**	第 2.2 節の提案
状態の直列化	**未**	移動にはアクターのフィールドを運ぶ必要がある
リース機構	**未**	`act` の排他移譲
配置決定器	**未**	効果 + 容量から配置を解く

コード移動が既にあることは大きい。移動の難所は通常コード配布であり、そこは AIPL → C JIT と `/actor/load` で解決済みである。本設計では**この移動機構が冗長化そのものを担う**ので、「あると便利な機能」ではなく**可用性の要**になる。したがって `/actor/load` の所要時間は**最初に測るべき数字**である（フェイルオーバ時間に直接効く）。

6.2 移動の契機と手順

契機	手順
ノード故障	リース期限切れ → **余裕のあるノードへ転送**して復帰
負荷偏り	純粋アクターを空きノードへ複製し、要求を分散。**状態が無いので停止不要**
RT 締切超過	期限に間に合っていないアクターを、余裕のあるノードへ移す
熱・電源	同上

効果集合が移動の難易度をそのまま決める。純粋アクターは複製するだけでよく、停止も同期も要らない。`mut` を持つものは状態を運ぶ必要がある。`act` を持つものはリース移譲を伴い、最も高くつく。**したがって移動の計画は、効果の軽いものから順に動かす**のが合理的である。

7　リアルタイム性

「全体として効率の良いリアルタイム OS」にするために、どのアクターが実時間制約を持つかを明確にする。

アクター	周期	締切	置き場所
`ServoMotor` ×6	10 Hz	硬	**RT スケジューラを持つノード**。縮退時は 1 台が 2 個
`ServoSensor` ×6	10 Hz	軟	CG 側の soft-Xinu
`ArmMover`	10 Hz	軟	任意
`Kinematics`	要求時	軟	任意（複製可）
`RecognitionActor`	撮像時	軟	重みを持つノード
`SafetyActor`	事象時	**硬**	記憶装置を持つノード

硬い締切を持つのは `ServoMotor` と `SafetyActor` だけである。**Pi 4 のプリエンプティブ優先度スケジューラ（実測 max jitter 10 μs、100 回中ミス 0）はこの用途に十分な性能を持つ**。残りは軟らかいので配置の自由度が高く、移動の対象にしやすい。

ただし現状 RT スケジューラは Pi 4 のみで実用段階にある。 Pi 5 は移植の途中（Stage 1–2）、Pi 3 は upstream のスケジューラである。6 台構成では**どのノードが硬い締切を引き受けられるかが不均一**であり、配置決定器はこれを制約として扱う必要がある。

8　学習の位置づけ

学習も「効果を持つアクターの 1 つ」として同じ枠組みで扱う。`Learner` は `!{ai, net}`（重みを読むなら `fs` も）であり、配置規則は第 2.3 節の表がそのまま適用される — 重みを持つノードならどこでもよく、複製も可能である。

現時点の推奨は Mac に置くことだが、これは設計ではなく性能の問題である。 Xinu は 3 ボードとも **D-cache が OFF**（アセスメント項目 10、最大の性能レバーが未使用）で、ベクトル化も数値ライブラリも無い。6 台 24 コアを足しても素の Mac に勝てる見込みが薄い。**D-cache を有効化した時点で再評価すべきであり、そのときは配置決定器に `Learner` を渡すだけで済む** — 枠組みを変える必要は無い。ここが本設計の利点である。

推論（`RecognitionActor`、TinyML 192→12→3 = 約 2.4k 積和）は最初からメッシュ側に置く。ロボットの制御経路上にあり、計算量も小さい。

9　実装の順序

依存関係の順に並べる。各段は前段が無いと意味を持たない。

段	内容	検証
1	act 効果の分離(Arm_serial_servo_write を mut から移す)	**完了**: `--type-check` が駆動する 3 メソッドだけを検出し、`ServoMotor` と `ShelfActor` が型で分かれた
2	配置規則の実装(効果集合→配置可能ノード集合)	単体テスト
3	状態の直列化(アクターのフィールドを運ぶ)	移動前後で状態が一致
4	リース機構(act の排他移	分断注入で二重駆動が起きないこと

第 1 段は今日から着手でき、費用がほぼゼロで効果が大きい。型検査だけで検証できる（実機不要）。第 5 段は現在の 3 台で実施でき、そこで取れるサイクル時間とジッタが、さらに台数を増やす価値を判断する実数になる。

10　着手前に潰すべき前提

1. **Pi 3 の 14 % 失敗。**原因特定済（単一 accept の HTTP サーバ）、修正はコミット済み。デプロイして効果を実測する。
2. **Pi 4 の 130 ms。**有線で Pi 5 の 3.6 倍。原因未究明。
3. **時刻同期が無い。**リースの期限判定はローカル時計で足りるが、経験 (s, a, r) をノード横断で突き合わせるには共通時刻が要る。
4. **D-cache が 3 ボードとも OFF。**学習の配置判断はこれを直してから。
5. **RT スケジューラが Pi 4 のみ。**6 台の能力が不均一。ただし異種混成は「能力の異なるノード群への配置問題」として**研究上の資産でもある。**

11　本書の限界

- レイテンシは 200 サンプル × 3 台、1 回の測定である。
- リースの T_r / T_e は実測レイテンシからの**見積もり**であり、分断注入試験で検証していない。
- `act` 分離は**実装済**（aios-claude `25b7c02`、aipl-line-simulator `bad8ab9`）。`--type-check` は `[type] no issues`、`test/aipl_sync.mjs` も申告漏れ 0 件。
- 転送方式は `act` が場所に縛られないことに依存する。スマートジョイント構成へ移行すると**この前提が崩れ、待機機方式へ戻す必要がある。**
- 縮退運転（1 台 2 軸）が実時間予算に収まるかは**仮定**であり、未測定。
- `/actor/load` は**実測済**（Pi 5 37.9 ms / Pi 4 114.8 ms、各 20 回）。ただしリースの T_r / T_e は依然として見積もりで、分断注入試験をしていない。
- 台数効果は**外挿**であり、実測は 3 台 1.87 倍（効率 62 %）のみ。
- アーム内蔵 Pi 5 に ROS が載っているという前提は**未確認。**

実測は *Raspberry Pi 3 B+ / 4 / 5* 上の *Xinu*、*MANET* は *WiFi ad-hoc (IBSS)*、*SSID* `MANET`、*ch 6*、`10.0.0.2/3/5`。アクター定義は `~/aipl_line_simulator/aipl/dofbot_xinu.abcl`。