資料—狀態—事件—行動：程式系統的四元動力結構

Data, State, Event, and Action: A Fourfold Dynamic Structure of Program Systems

v0.1 / 2026-07-27 / Neo.K with Aletheia / 初版完成

https://thisoneisneok.com/html/papers/pu-1-03.html

SECTION

Data, State, Event, and Action: A Fourfold Dynamic Structure of Program Systems

論文編號： PU-1-03

系列：「程式宇宙書系」第 1 冊《系統性程式設計》

作者： Neo.K with Aletheia

機構： EveMissLab／一言諾科技有限公司

版本： v0.1

日期： 2026 年 7 月 27 日

SECTION

摘要

前篇建立了問題世界的實體、關係、規則與系統邊界，但靜態世界模型仍不足以描述程式如何隨時間運作。任何持續系統都必須回答：世界目前是什麼狀態、什麼事情發生了、誰想做什麼、哪些資料只是觀測或投影，以及一次變化如何被判定為合法。

本文提出程式系統的四元動力結構：

【mathbb D_P
=
<
D,
S,
E,
A
>】

其中：

- D：Data，資料；
- S：State，狀態；
- E：Event，事件；
- A：Action，行動。

本文核心命題是：

【D
≠
S
≠
E
≠
A】

資料是對世界、狀態、事件或主體主張的表示與證據；狀態是系統在特定時間對世界合法情況的權威判定；事件是某項已發生變化或事實的不可撤回主張；行動則是某個主體希望系統執行之狀態變更請求。四者相互關聯，但不應互相替代。

本文將一次完整狀態變化形式化為：

A⊠
xrightarrowValidate
A⊠^(v)
xrightarrowAuthorize

其中：

- 行動先被驗證；
- 再被授權；
- 執行後形成事件；
- 事件改變權威狀態；
- 狀態與歷史再被投影為查詢、畫面、報表與資料。

此模型解釋了為何：

- 使用者點擊按鈕不等於事件已發生；
- API 收到請求不等於行動已成功；
- 日誌中出現一行不等於權威事件成立；
- 資料表中的值不一定是當前權威狀態；
- success=true 不一定代表外部世界已完成。

本文提出狀態判準：

> State⌧(x)
> ⇔
> Authoritative(x)
> ∧
> Current(x,t)
> ∧
> InvariantValid(x)

事件判準：

> Event(e)
> ⇔
> Occurred(e)
> ∧
> Identified(e)

以及行動判準：

$$
\begin{aligned}
&\mathrm{Action}(a)\\
&\Leftrightarrow\\
&\mathrm{Requested}(a)\\
&\wedge\\
&\mathrm{ActorKnown}(a)\\
&\wedge\\
&\mathrm{TargetKnown}(a)\\
&\wedge\\
&\mathrm{ExpectedEffectKnown}(a)
\end{aligned}
$$

本文區分：

- 命令／請求與事件；
- 當前狀態與歷史事實；
- 權威狀態與資料投影；
- 世界事件與處理事件；
- 意圖完成與狀態轉移成功。

本文進一步建立狀態轉移函數：

$$
\begin{aligned}
&S_{t+1}\\
&=\\
&\delta\\
&(\\
&S_t,\\
&E_{t+1},\\
&R_t\\
&)
\end{aligned}
$$

其中 R_t 為當時有效的規則與不變量。事件本身不直接等於新狀態；它必須在規則下被解釋與套用。

本文也處理長時程與分散式系統中的關鍵問題：

- 事件重複；
- 訊息亂序；
- 延遲到達；
- 並行競爭；
- 部分成功；
- 重試；
- 冪等；
- 補償；
- 快照與重建；
- 外部世界確認；
- 事件時間與處理時間分離。

本文提出：

> 【Retry
> not⇒
> RepeatEffect】

成熟行動必須具備身份、冪等鍵、前置條件、授權、預期效果、結果證據與失敗語意。否則重試就可能把同一意圖轉化為多次世界效果。

本文使用待辦事項、訂單付款、網站部署、醫療處置、AI Agent 與研究進度等案例，說明四元結構如何辨識常見錯誤。本文最後提出可證偽研究綱領，包括資料—狀態混淆率、命令—事件混淆率、重複效果率、事件重建一致性、權威狀態辨識、狀態機完整性、處理時間偏差與外部世界確認率。

本文為下一篇〈責任、模組與契約〉建立動力地基：當資料、狀態、事件與行動被分離後，才能進一步決定哪個模組擁有狀態、誰負責處理事件、哪些行動必須經過契約，以及跨邊界失敗如何被治理。

關鍵詞： 資料、狀態、事件、行動、狀態轉移、事件溯源、冪等、分散式系統、程式動力學

SECTION

Abstract

This paper develops a fourfold dynamic structure of program systems:

> mathbb D_P
> =
> <
> D,
> S,
> E,
> A
> >

Data represents observations, records, or projections. State is the authoritative current condition of a problem world. Events assert that something has occurred. Actions are requests by actors to produce state changes.

The core thesis is:

> D
> ≠
> S
> ≠
> E
> ≠
> A

A valid transition is modeled as:

> A⊠
> →
> A⊠^(v)
> →
> A⊠^(a)
> →

The paper distinguishes commands from events, historical facts from current state, authoritative state from projections, and computation completion from world completion. It addresses idempotency, retries, duplication, ordering, concurrency, partial success, compensation, snapshots, and external confirmation.

Keywords: data, state, event, action, state transition, idempotency, event sourcing, distributed systems

很多程式從資料表開始。

設計者建立：

users
orders
tasks
payments
然後加入 CRUD。

但 CRUD 只能說明資料可以被建立、讀取、修改與刪除，不能完整回答：

- 這些資料是否是權威狀態；
- 誰有權修改；
- 什麼事件使它改變；
- 修改是否代表現實已發生；
- 重複請求是否會重複造成效果；
- 失敗後世界停在哪裡；
- 歷史如何被保留。

  真正的系統不是資料倉庫，而是時間中的狀態轉換結構。

  本文定義：

> 【mathbb D_P
> =
> <
> D,
> S,
> E,
> A

SECTION

2.1 Data

資料是被記錄、傳輸、儲存或計算的表示。

它可能代表：

- 感測值；
- 使用者輸入；
- 狀態投影；
- 歷史事件；
- 快取；
- 預測；
- 主張；
- 證據。

SECTION

2.2 State

狀態是問題世界在特定時間的權威合法配置。

SECTION

2.3 Event

事件是「某件事已發生」的識別化事實主張。

SECTION

2.4 Action

行動是由主體提出、希望改變世界的請求或操作意圖。

SECTION

3.1 資料不等於狀態

同一狀態可能有多份資料投影。

同一資料也可能：

- 過時；
- 不完整；
- 未驗證；
- 只是快取；
- 只是使用者主張。

SECTION

3.2 行動不等於事件

「請付款」是行動。

「付款已被授權」才可能是事件。

SECTION

3.3 事件不等於狀態

「收到付款授權」事件發生後，訂單是否進入 paid，仍取決於規則與當前狀態。

SECTION

3.4 狀態不等於歷史

當前 paid 狀態不能說明：

- 何時付款；
- 誰付款；
- 是否重試；
- 是否曾失敗；
- 是否經過爭議。

SECTION

3.5 日誌不等於事件

日誌可能包含：

- 除錯訊息；
- 重複輸出；
- 處理嘗試；
- 例外堆疊；
- 非權威敘述。

事件則需要身份、因果位置與語意。

資料在程式系統中不是單一種類。

SECTION

4.1 輸入資料

由使用者、感測器或外部服務提供。

輸入資料通常只是主張，尚未成為權威狀態。

SECTION

4.2 權威資料

被系統指定為狀態判定依據的資料。

例如：

- 主資料庫；
- 事件歷史；
- 經簽章的外部回覆；
- 已核准文件。

SECTION

4.3 投影資料

為查詢、畫面、搜尋或報表建立的衍生表示。

投影可被重建，不必是權威來源。

SECTION

4.4 快取資料

為效能保存的暫時副本。

快取可能過時，因此：

> Cache(x)
> not⇒
> CurrentState(x)

SECTION

4.5 證據資料

用來證明事件、決策與世界效果的材料，例如：

- 簽章；

- 回執；
- 影像；
- 測試結果；
- 操作紀錄；
- 外部確認。

SECTION

4.6 資料分類必須可見

成熟系統應能回答：

這份資料是輸入、權威狀態、投影、快取，還是證據？
否則開發者容易對錯誤資料來源做出世界判定。

本文提出：

> State⊠(x)
> ⇔
> Authoritative(x)
> ∧
> Current(x,t)
> ∧
> InvariantValid(x)

SECTION

5.1 權威性

必須知道哪一結構有權決定當前狀態。

SECTION

5.2 當前性

狀態具有時間。

昨天的正確值，不一定是今天的狀態。

SECTION

5.3 合法性

狀態必須符合世界規則與不變量。

SECTION

5.4 局部狀態與全域狀態

分散式系統中，單一節點可能只擁有局部狀態：

> S□^((i))
> ⊆
> S□^(global)

但全域狀態可能無法在同一時刻被單一節點完整看見。

SECTION

5.5 暫時不一致

兩個合法節點可能短暫持有不同版本：

> S□^((1))
> ≠
> S□^((2))

這不一定代表資料損壞，但必須有：

- 版本；
- 合併；

- 衝突；
- 一致性模型；
- 最終判定。

本文定義事件：

> E
> =
> <
> id,
> type,
> actor,
> subject,
> time,
> cause,
> payload,
> evidence,
> version
> >

SECTION

6.1 事件身份

每個事件需要唯一或可去重身份。

SECTION

6.2 事件類型

事件名稱應以已發生形式表達：

PaymentAuthorized
TaskCompleted
DeploymentStarted
ReviewRejected

而不是：

AuthorizePayment
CompleteTask
StartDeployment
RejectReview
後者較像命令或行動。

SECTION

6.3 行動者與對象

需要知道誰或什麼造成事件，以及事件作用於哪個實體。

SECTION

6.4 事件時間

至少可區分：

- 發生時間；
- 記錄時間；
- 接收時間；
- 處理時間。

SECTION

6.5 因果來源

事件應能連接：

- 原始命令；
- 前一事件；
- 工作流；

- 外部證據；
- 規則版本。

SECTION

6.6 事件不可任意改寫

若事件代表已發生事實，錯誤修正應以新事件完成，而不是靜默改寫舊歷史。

SECTION

7.1 事件需要領域意義

記憶中的位元翻轉不必成為領域事件。

SECTION

7.2 處理事件

系統內部可能記錄：

- MessageReceived；
- RetryScheduled；
- CacheRebuilt。

這些是處理事件，不一定是世界事件。

SECTION

7.3 世界事件

世界事件代表問題世界中的有效變化，例如：

- 訂單已付款；
- 任務被取消；

- Agent 任務已被撤回。

SECTION

7.4 外部事件

外部世界發生，系統只能觀測：

- 網路中斷；
- 使用者離線；
- 物流到達；
- 法規更新。

SECTION

7.5 事件層級

系統應區分：

```
event_scope:
 processing: true
 system: true
 domain: false
 physical_world: false
```

避免把內部處理成功冒充領域事件。

本文定義：

> A
> =
> <
> id,
> actor,
> intent,
> target,
> precondition,
> permission,

SECTION

8.1 行動身份

每一重要行動應可追蹤與去重。

SECTION

8.2 行動者

必須知道：

- 誰要求；
- 誰批准；
- 誰執行；
- 誰承擔責任。

SECTION

8.3 意圖

行動不只包含操作名稱，也應保留其目的。

SECTION

8.4 目標

行動作用於哪個世界實體或狀態。

SECTION

8.5 前置條件

只有特定世界狀態下才可執行。

SECTION

8.6 權限

前置條件成立，不表示行動者有權執行。

SECTION

8.7 預期效果

應明示預期產生：

- 哪些事件；
- 哪些狀態差分；
- 哪些外部效果；
- 哪些證據。

SECTION

8.8 期限與過期

過期行動可能不應繼續執行。

SECTION

8.9 冪等性

重複提交同一行動，不應產生多次世界效果。

SECTION

8.10 失敗語意

需區分：

- 可重試；
- 不可重試；
- 已部分成功；
- 需要補償；
- 需要人類判定。

  完整流程為：

> A□
> xrightarrowValidate
> A□^(v)
> xrightarrowAuthorize
> A□^(a)
> xrightarrowExecute
> E_(t+1)

SECTION

9.1 驗證

檢查：

- 結構；
- 目標；
- 前置條件；
- 時效；
- 世界版本。

SECTION

9.2 授權

檢查行動者是否具有合法權力。

SECTION

9.3 執行

執行可能：

- 成功；
- 失敗；
- 部分成功；
- 超時；
- 結果未知。

SECTION

9.4 事件形成

只有被系統或世界接受為已發生事實的結果，才應形成領域事件。

SECTION

9.5 拒絕也是結果

拒絕行動可形成：

ActionRejected
AuthorizationDenied
PreconditionFailed
但這些事件不代表預期世界效果發生。

事件經規則套用後形成新狀態：

S_(t+1)
=
δ
(

SECTION

10.1 事件不自動改變狀態

同一事件在不同狀態下可能：

- 被接受；
- 被忽略；
- 形成衝突；
- 進入爭議；
- 需要人工覆核。

SECTION

10.2 規則版本

事件的解釋依賴當時有效規則。

> S_(t+1)
> =
> δ_(R□)
> (
> S□,
> E_(t+1)
>)

若規則後續改變，不能任意用新規則重寫舊事件意義。

SECTION

10.3 不變量檢查

套用後必須保持：

$$Inv(S_{t+1})=true$$

若不變量失敗，系統可能：

- 拒絕事件；
- 將事件標記為無法套用；
- 進入錯誤狀態；
- 觸發補償；
- 需要人工介入。

SECTION

10.4 派生狀態

某些狀態可由事件歷史重建：

$$S_t = Fold(S_0, E_1, E_2, \dots, E_t)$$

SECTION

10.5 快照

為效能可保存：

$$
\text{Snapshot}_k = S_k
$$

之後只需套用 k 之後的事件。

快照是狀態加速器，不是歷史替代品。

狀態與歷史會被投影為：

- 查詢資料；
- UI 畫面；
- 搜尋索引；
- 報表；
- 通知；
- API 回覆；
- 分析指標。

形式上：

$$
D_k^{(i)} = \pi_k(S_k, H_k)
$$

SECTION

11.1 多投影

同一權威狀態可有多個投影。

例如訂單狀態可投影成：

- 客戶畫面；
- 商家後台；
- 物流工作單；
- 財務報表；
- 稽核紀錄。

SECTION

11.2 投影延遲

投影可能晚於權威狀態。

因此 UI 尚未更新，不代表狀態沒有改變。

反之，樂觀 UI 顯示成功，也不代表權威狀態已接受。

SECTION

11.3 投影可重建

若投影損壞，應能從權威狀態與歷史重建。

SECTION

11.4 投影不應反向僭位

查詢資料通常不應被直接當作權威來源修改。

否則：

- 投影與狀態混合；
- 責任不清；

- 事件歷史失真；
- 同步困難。

SECTION

12.1 發生時間

世界中事情真正發生的時間。

SECTION

12.2 接收時間

系統收到事件的時間。

SECTION

12.3 處理時間

系統實際處理的時間。

SECTION

12.4 生效時間

事件在制度或狀態上開始有效的時間。

因此：

> $T_{occur} \neq T_{receive} \neq T_{process} \neq T_{effective}$

SECTION

12.5 延遲事件

某事件可能晚到，但仍屬於較早世界歷史。

SECTION

12.6 未來生效

某規則或委任可以先被記錄，但在未來時間才生效。

SECTION

12.7 時間倒置風險

若只使用處理時間排序，可能把真正先發生的事件放在後面。

SECTION

13.1 全域順序未必存在

分散式系統中，不同節點可能看到不同事件順序。

SECTION

13.2 因果順序

若事件 e□ 依賴 e□：

> e□
> prec
> e□

則應保留因果關係。

SECTION

13.3 無關事件

沒有因果關係的事件可以並行。

SECTION

13.4 亂序處理

晚到事件可採：

- 暫存；
- 重排；
- 版本檢查；
- 補償；
- 人工審核；
- 接受為新主張但不重寫歷史。

SECTION

13.5 事件序號不是因果全部

自增 ID 可以提供資料庫順序，但不必等於世界因果順序。

SECTION

14.1 同時行動

兩個主體可能同時對同一實體提出行動。

例如兩人同時購買最後一件商品。

SECTION

14.2 樂觀並行

行動攜帶所依據的狀態版本：

> A
> =
> (
> target,
> expectedVersion
>)

若版本已改變，行動需重新評估。

SECTION

14.3 悲觀鎖定

先取得排他控制，再執行。

SECTION

14.4 衝突不是技術例外全部

某些衝突是問題世界中的正常狀態，例如多人同時編輯、競標或資源競爭。

SECTION

14.5 合併規則

合併需要明示：

- 可自動合併；
- 最後寫入者；

- 優先角色；
- 人工解決；
- 形成爭議狀態。

SECTION

15.1 重試的必要性

網路與外部服務可能：

- 超時；
- 中斷；
- 回覆遺失；
- 處理成功但確認失敗。

SECTION

15.2 重試風險

同一付款行動若被處理兩次，可能造成重複扣款。

SECTION

15.3 冪等鍵

> IdempotencyKey(A)=k

系統若已處理 k，重複請求應回傳原結果，而非再次產生效果。

SECTION

15.4 冪等不是忽略所有重複

同一內容但不同真實意圖，可能是兩個合法行動。

因此身份應來自行動實例，而不是只比較 payload。

SECTION

15.5 核心命題

> 【Retry
> not⇒
> RepeatEffect】

SECTION

16.1 原子操作的限制

跨多個系統的世界變化，通常無法由單一交易完全包住。

SECTION

16.2 部分成功

例如：

付款已完成
庫存更新失敗
通知未送出
此時不能簡單回覆成功或失敗。

SECTION

16.3 補償

補償是新的行動與事件，不是讓歷史未曾發生。

例如：

- 退款；

- 釋放庫存；
- 撤回發布；
- 修正文件；
- 恢復權限。

SECTION

16.4 補償失敗

補償本身也可能失敗，因此需要：

- 狀態；
- 重試；
- 人工接管；
- 證據；
- 時限。

SECTION

16.5 長時交易

長時工作流應被建模為多階段狀態機，而不是一個長函式。

SECTION

17.1 內部事件不等於外部完成

系統可以產生：

```
EmailRequestAccepted
ShipmentDispatched
DeviceCommandSent
```

但這些事件不等於：

EmailRead
ShipmentReceived
DevicePhysicallyChanged

SECTION

17.2 證據分層

可區分：

- 請求已建立；
- 外部系統已接受；
- 外部系統宣稱已完成；
- 本系統已觀測到效果；
- 受影響主體已接受結果。

SECTION

17.3 不可觀測效果

若系統無法直接驗證，狀態應保留：

pending-external-confirmation
externally-claimed
unverified
contested

SECTION

17.4 完成條件

> Complete
> =
> InternalState
> ∧
> ExternalEvidence
> ∧

不同任務所需層級不同，但不能默認內部事件等於世界完成。

SECTION

18.1 事件溯源

事件歷史作為權威來源：

> H⊠
> =
> [
> E⊠,E⊠,…,E⊠
>]

狀態由歷史折疊得到。

SECTION

18.2 優點

- 歷史可追蹤；
- 狀態可重建；
- 規則變化可分析；
- 投影可重新生成；
- 責任與因果較清楚。

SECTION

18.3 代價

- 事件 schema 演化；
- 歷史資料量；

- 重建成本；
- 錯誤事件修正；
- 隱私與刪除；
- 規則版本管理。

SECTION

18.4 狀態儲存

直接保存當前狀態較簡單，但容易遺失：

- 變化原因；
- 中間過程；
- 失敗；
- 撤回；
- 責任證據。

SECTION

18.5 混合模式

多數系統可使用：

- 權威當前狀態；
- 關鍵領域事件；
- 稽核歷史；
- 查詢投影；

而不必將每個位元變化都事件化。

SECTION

19.1 歷史保存不是無限保存

事件可能包含：

- 個資；
- 敏感內容；
- 商業機密；
- 錯誤資料。

SECTION

19.2 刪除與可稽核衝突

需區分：

- 事件存在；
- payload 可見；
- 身分可識別；
- 證據可驗證；
- 法律保留期。

SECTION

19.3 匿名化與密鑰銷毀

可用：

- 資料最小化；
- 欄位加密；

- 身分替換；
- 密鑰銷毀；
- 衍生投影刪除；

兼顧歷史完整與隱私。

SECTION

19.4 錯誤修正

不應直接竄改舊事件，而可新增：

EventCorrected
ClaimWithdrawn
IdentityRedacted
保留修正本身的歷史。

SECTION

20.1 行動

CompleteTask
包含：

- 行動者；
- 任務 ID；
- 依據版本；
- 完成證據；
- 冪等鍵。

SECTION

20.2 事件

TaskCompletionClaimed

不一定直接是 TaskCompleted。

若需要審核，還可能產生：

TaskCompletionAccepted
TaskCompletionDisputed

SECTION

20.3 狀態

active
completion-pending
completed
disputed

SECTION

20.4 資料

UI 上的勾選框只是狀態投影，不是完整歷史與證據。

SECTION

21.1 行動

客戶提出付款請求。

SECTION

21.2 外部事件

付款服務回覆：

PaymentAuthorized

SECTION

21.3 系統事件

本地成功處理後：

OrderPaymentRecorded

SECTION

21.4 狀態

訂單進入 paid，但最終銀行結算可能仍未完成。

SECTION

21.5 重複與亂序

系統必須處理：

- 回呼重複；
- 取消事件先到；
- 授權晚到；
- 退款與付款交錯；
- 本地更新失敗。

SECTION

21.6 完成分層

```
payment:
 provider_authorization: "confirmed"
 local_order_state: "paid"
 bank_settlement: "pending"
 customer_acceptance: "not-required"
```

SECTION

22.1 行動

DeployRelease

SECTION

22.2 事件

- DeploymentRequested；
- ArtifactUploaded；
- ProcessStarted；
- HealthCheckPassed；
- TrafficShifted；
- DeploymentFailed。

SECTION

22.3 狀態

planned
approved
deploying
verifying
active
degraded
rolled-back
failed

SECTION

22.4 資料

部署頁面、日誌、監控指標與版本標籤是不同投影。

SECTION

22.5 世界完成

Health check 通過不代表所有使用者流程正常。

還可能需要：

- 真實流量；
- 錯誤率；

- 人工驗收；
- 外部 DNS；
- 商業指標。

SECTION

23.1 行動

醫師開立處置，或病人同意某項治療。

SECTION

23.2 事件

需區分：

- TreatmentRecommended；
- ConsentRecorded；
- TreatmentStarted；
- TreatmentAdministered；
- OutcomeObserved。

SECTION

23.3 狀態

醫療計畫與病人身體狀態不是同一狀態空間。

SECTION

23.4 資料

檢驗值與病人回報都是資料，需要：

- 來源；
- 時間；
- 品質；
- 適用範圍；
- 專業解釋。

SECTION

23.5 不可逆性

高風險行動不能因資料欄位完整就自動執行，仍需保留同意與專業責任。

SECTION

24.1 行動

Agent 提出：

ModifyFile
CallAPI
PublishWebsite
SendMessage

SECTION

24.2 工具回覆

工具回傳只是處理資料，不必直接成為領域事件。

SECTION

24.3 事件形成

需經驗證後才形成：

FileModified
WebsitePublished
MessageAcceptedByProvider

SECTION

24.4 狀態

Agent 自身也有：

- 目標狀態；
- 計畫狀態；
- 權限狀態；
- 預算狀態；
- 檢查點；
- 等待人類狀態。

SECTION

24.5 重試

Agent 若不知道工具已成功，重試可能造成：

- 重複寄信；
- 重複付款；
- 重複發布；
- 重複刪除。

所以 Agent Runtime 必須內建行動身份與冪等證據。

SECTION

25.1 行動

ProposeHypothesis
RunExperiment
WritePaper
ClaimResult

SECTION

25.2 事件

- HypothesisRegistered；
- ExperimentExecuted；
- EvidenceProduced；
- CounterexampleFound；
- ManuscriptCompleted；
- ClaimVerified。

SECTION

25.3 狀態

論文完成與命題驗證是不同狀態。

drafted
reviewed
partially-supported
contradicted
verified
open

SECTION

25.4 資料

實驗輸出、文獻摘要與模型推論是證據資料，不自動等於研究事實。

SECTION

25.5 失敗也是事件

失敗計算與反例是研究世界的重要事件，不應只留在臨時日誌。

- 資料即狀態： 任意資料值被當成當前權威世界。
- 快取即權威： 過時投影被用於重大決策。
- 命令即事件： 收到請求就宣稱事情已發生。
- 日誌即歷史： 除錯輸出被當成領域事件證據。
- 事件即狀態： 忽略規則、前置狀態與不變量。
- 狀態即歷史： 只保存當前值，無法追蹤原因與責任。
- 內部事件即外部完成： 系統處理成功被當成物理效果成功。
- 單一時間： 發生、接收、處理與生效時間混為一談。
- 資料庫順序即因果： 自增 ID 被誤認為世界先後。
- 重試即重做： 相同意圖造成多次世界效果。
- payload 去重： 內容相同的不同合法行動被錯誤合併。
- 所有錯誤皆可重試： 永久錯誤與權限拒絕被無限重試。
- 部分成功二元化： 跨系統工作流只回傳成功或失敗。
- 補償即回滾： 忽略補償是新的歷史事件。
- 投影反向寫入： 查詢模型直接僭位成權威狀態。
- 事件過度化： 每個技術細節都變成領域事件。
- 事件不足： 重要責任與世界變化只留在可覆寫欄位。
- 規則版本遺忘： 用新規則重新解釋舊歷史。
- 未知結果強制成功： 超時後直接假定外部操作完成。
- 完成狀態單層化： 計算、系統、外部與意圖完成混合。
- 事件永久保存迷思： 忽略隱私、刪除與資料最小化。

- Agent 工具回覆僭位： 工具文字被直接當成真實世界事件。

SECTION

27.1 資料—狀態混淆率

從事故與錯誤決策中統計，多少源於使用：

- 快取；
- 投影；
- 未驗證輸入；
- 過時副本；

作為權威狀態。

SECTION

27.2 命令—事件混淆率

統計收到請求、排入佇列或開始處理後，系統過早宣稱完成的比例。

SECTION

27.3 重複效果率

> R_D
> =
> (|duplicate world effects|)/(|retried actions|)

測量冪等機制的實際效果。

SECTION

27.4 事件重建一致性

從事件歷史重建狀態，與權威快照比較：

> C_R
> =
> 1-
> d
> (
> S_(replayed),
> S_(authoritative)
>)

SECTION

27.5 事件語意完整率

測量重要事件是否具有：

- 身份；
- 主體；
- 對象；
- 發生時間；
- 因果；
- 證據；
- 規則版本。

SECTION

27.6 權威狀態辨識率

檢查開發者與維護者能否正確指出各種資料來源中的權威狀態。

SECTION

27.7 處理時間偏差

比較依處理時間排序與依事件時間、因果關係排序後的業務結果差異。

SECTION

27.8 亂序處理成功率

測量系統對晚到、重複與交錯事件的正確狀態維持能力。

SECTION

27.9 部分成功可見率

統計跨系統工作流中，部分成功是否被正式表示，而非壓成二元結果。

SECTION

27.10 外部世界確認率

測量系統宣稱完成的行動中，有多少具有足夠外部證據。

SECTION

27.11 快照—歷史偏差

比較快照與完整事件歷史在稽核、除錯與責任判定上的資訊差異。

SECTION

27.12 四元建模教學實驗

比較先學 CRUD 與先學資料—狀態—事件—行動區分的學習者，在分散式錯誤、重試與狀態設計上的表現。

- 資料、狀態、事件與行動是不同的系統存在。

- 資料可以是輸入、權威資料、投影、快取或證據。
- 狀態是特定時間下權威、當前且符合不變量的世界配置。
- 行動是改變世界的請求，不是已發生事實。
- 事件是已發生事實的識別化主張，不是未來命令。
- 日誌不自動具有領域事件地位。
- 事件需經規則套用才能形成新狀態。
- 當前狀態無法取代事件歷史。
- 同一狀態可以具有多個不同角色投影。
- 投影延遲不等於權威狀態尚未改變。
- 發生、接收、處理與生效時間必須分離。
- 資料庫順序不等於完整因果順序。
- 並行衝突可能是正常世界狀態，而非單純技術例外。

14.

> Retry
> not⇒
> RepeatEffect

- 冪等身份應綁定行動實例，而非只比較資料內容。
- 部分成功必須成為正式狀態。
- 補償是新的行動與事件，而不是真正刪除歷史。
- 內部處理完成不等於外部世界完成。
- 事件歷史、當前狀態、快照與投影各有不同責任。
- 高風險行動需要外部證據與必要的人類接受。

- Agent Runtime 必須保存行動身份、工具結果、事件證據與檢查點。

22.

> 【程式系統的動力，】

> 【不是資料被任意修改，】

> 【而是有身份的行動在規則下形成事件，】

> 【事件再使合法世界狀態持續演化。】

SECTION

29.1 承接 PU-1-01

PU-1-01 將程式定義為可執行問題模型。

本篇把其中的「可執行」展開為：

- 行動驗證；
- 授權；
- 事件形成；
- 狀態轉移；
- 資料投影。

SECTION

29.2 承接 PU-1-02

PU-1-02 建立了實體、關係、規則與邊界。

本篇讓這些靜態結構進入時間：

- 實體持有狀態；
- 關係經事件建立與終止；
- 規則控制轉移；
- 邊界決定外部證據與完成程度。

SECTION

29.3 銜接 PU-1-04

下一篇將回答：

- 哪個模組擁有哪個狀態；
- 誰有責任處理哪類事件；
- 哪些行動能跨模組；
- 模組之間如何以契約交換資料、命令、事件與證據；
- 為何責任邊界不等於檔案或服務邊界。

當設計者只看見資料，程式就像一組讀寫欄位的技巧。

但持續系統真正需要維持的是：

- 世界目前處於什麼狀態；
- 哪個主體希望做什麼；
- 該行動是否有效與有權；
- 哪件事情真正發生；
- 事件如何改變世界；
- 哪份資料只是畫面或投影；
- 失敗與重試會不會重複傷害世界；

- 外部效果是否真的完成。

  因此，CRUD 並不是錯誤。

  它只是四元動力結構在儲存層的一種低階投影。

  本文將系統動力收束為：

> 【Action
> →
> Validation
> →
> Authorization
> →
> Event
> →
> State
> →
> Data Projection】

更完整地說：

> 【Program Dynamics
> =
> Intentional Action
> +
> Rule-Governed Event
> +
> Authoritative State
> +
> Evidence-Bearing Data】

這個結構使系統能回答：

誰做了什麼？
依據哪個世界狀態？
使用哪條規則？
真正發生了什麼？

世界因此變成什麼？
我們如何知道？
若只完成一部分，現在應該怎麼辦？

本文的最終命題是：

> 【資料可以被覆寫，】

> 【但世界變化必須有事件、規則、責任與證據。】

只有如此，程式才能從資料處理器成為可理解、可恢復、可稽核與可治理的持續世界系統。

```
program_dynamics:
 action:
  action_id: "pay-order-001"
  actor: "customer-17"
  intent: "pay for order"
  target: "order-932"
  expected_version: 4
  idempotency_key: "customer17-order932-payment1"
 validation:
  preconditions:
   - "order.status == payment-pending"
  permission:
   - "customer owns order"
 event:
  event_id: "payment-authorized-889"
  type: "PaymentAuthorized"
  occurred_at: "2026-07-27T14:30:00+08:00"
  caused_by: "pay-order-001"
  evidence: "provider-receipt-332"
 state:
  before: "payment-pending"
  after: "paid"
  version: 5
 data_projections:
  customer_view: "Paid"
  merchant_queue: "Ready for fulfillment"
  finance_report: "Settlement pending"
data_object:
 name: "order-summary"
 role: "projection"
 authoritative: false
 generated_from:
  - "order-state"
```

```
    - "payment-events"
  latency_target: "5s"
  writable: false
event_time:
  occurred_at: "2026-07-27T14:29:50+08:00"
  received_at: "2026-07-27T14:30:01+08:00"
  processed_at: "2026-07-27T14:30:03+08:00"
  effective_at: "2026-07-27T14:29:50+08:00"
workflow_state:
  workflow_id: "deployment-42"
  status: "partially-complete"
  completed:
    - "artifact-uploaded"
    - "process-started"
  pending:
    - "external-dns-confirmation"
  failed:
    - "smoke-test-mobile"
  compensation:
    available: true
    action: "rollback-release"
  human_review_required: true
```

- PU-1-01　程式不等於程式碼：可執行問題模型的基本定義
- PU-1-02　問題世界建模：實體、關係、規則與系統邊界
- PU-1-03　資料—狀態—事件—行動：程式系統的四元動力結構
- PU-1-04　責任、模組與契約：從檔案分類到系統邊界
- PU-1-05　可視化作為外部結構記憶：多角色、多尺度的程式理解
- PU-1-06　失敗也是程式：驗證、可觀測、恢復與長期維護

SECTION

Neo.K／EveMissLab 相關理論

- Neo.K with Aletheia，《程式不等於程式碼：可執行問題模型的基本定義》，2026。
- Neo.K with Aletheia，《問題世界建模：實體、關係、規則與系統邊界》，2026。

- Neo.K with Aletheia，《三宇宙耦合動力學：從想要、表示到世界改變》，2026。
- Neo.K with Aletheia，《數位宇宙作為人工存在域：實體、狀態、規則與歷史》，2026。
- Neo.K with Aletheia，《Agent Runtime：能力規劃、工具調用與可恢復執行》，2026。
- Neo.K with Aletheia，《時間—空間程式控制：長時程 Agent 的迴圈、切片與反身執行》，2026。

SECTION

一般理論背景

- Harel, D., “Statecharts: A Visual Formalism for Complex Systems,” 1987.
- Lamport, L., “Time, Clocks, and the Ordering of Events in a Distributed System,” 1978.
- Gray, J. and Reuter, A., Transaction Processing, 1992.
- Fowler, M., Patterns of Enterprise Application Architecture, 2002.
- Evans, E., Domain-Driven Design, 2003.
- Kleppmann, M., Designing Data-Intensive Applications, 2017.
- Vernon, V., Implementing Domain-Driven Design, 2013.
- Richardson, C., Microservices Patterns, 2018.
- Helland, P., “Life Beyond Distributed Transactions,” 2007.
- Pat Helland, “Immutability Changes Everything,” 2015.
- Newman, S., Building Microservices, 2015.
- Hoare, C. A. R., “Communicating Sequential Processes,” 1978.

SECTION

v0.1 — 2026-07-27

- 完成十八篇地基論文第九篇。
- 建立資料、狀態、事件、行動四元動力模型。
- 明確區分輸入、權威、投影、快取與證據資料。
- 提出狀態、事件與行動的形式判準。
- 建立行動驗證、授權、執行、事件、狀態與投影鏈。
- 分析事件層級、規則版本、不變量與狀態重建。
- 區分發生、接收、處理與生效時間。
- 分析亂序、因果、並行與衝突。
- 建立重試與冪等原則。
- 正式建模部分成功、補償與長時工作流。
- 區分內部處理與外部世界確認。
- 分析事件溯源、快照、隱私與歷史修正。
- 加入待辦、付款、部署、醫療、AI Agent 與研究進度案例。
- 提出二十二類失敗模式與十二項可證偽研究方向。
- 完成與 PU-1-04《責任、模組與契約》的銜接。