# 基於 OpenWrt 之 prplOS 架構下利用 Ambiorix Trace Zone、WiFi FSM 與 FTA 覆寫機制進行架構視覺化與整合分析報告

## 1. 執行摘要

隨著寬頻網路設備（Customer Premises Equipment, CPE）功能的日益複雜化，傳統基於單一韌體映像檔的嵌入式系統架構已難以滿足現代電信營運商對於服務靈活性、模組化及快速部署的需求。在此背景下，prpl Foundation 推出的 prplOS 作業系統，基於成熟的 OpenWrt Linux 發行版，引入了 Ambiorix（amx）中介軟體框架，實現了從底層匯流排（Bus）到上層應用服務的全面解耦 1。然而，這種高度模組化與微服務化的架構轉變，雖然提升了系統的可擴展性與硬體無關性，卻也顯著增加了系統行為的可觀測性（Observability）難度。當一個 WiFi 連線請求穿越多個透過非同步信號（Signals）溝通的服務與外掛程式（Plugins），並最終觸發硬體抽象層（HAL）的操作時，傳統的日誌分析手段往往難以重建完整的執行脈絡。

本研究報告旨在深入探討並構建一套針對 prplOS 的高階架構視覺化分析方法論。透過整合 Ambiorix 框架內建的 libsahtrace 追蹤區域（Trace Zone）機制、WiFi 子系統（特別是 prplMesh 與 Wireless Daemon）的有限狀態機（Finite State Machine, FSM）狀態變遷邏輯，以及功能表抽象層（Function Table Abstraction, FTA）的可覆寫函式指標特性，我們能夠將靜態的程式碼邏輯轉化為動態的系統行為視圖。

本報告首先解構 Ambiorix 的核心運作機制，闡述 libamxp 的事件迴圈與信號/槽（Signal/Slot）模式如何構成系統的神經網絡，以及如何透過 libsahtrace 的動態配置來擷取關鍵的控制流資訊 3。接著，深入剖析 WiFi 核心邏輯中的 FSM 設計模式，特別是 Beerocks（prplMesh）中的 C++ 狀態機實作，並提出利用 FSM\_MOVE\_STATE 巨集作為狀態變遷追蹤點的策略 5。在硬體互動層面，報告詳細檢視了 FTA 結構體的設計原理，展示如何利用其函式指標（Function Pointers）的動態連結特性，透過攔截（Interception）技術植入非侵入式的效能探針，以精確測量硬體操作的延遲與副作用 6。

最終，本報告提出了一套整合性的資料處理管線，將上述異質性的追蹤數據標準化為 Perfetto 的 Trace Event 格式及 Mermaid 的狀態圖語法。這不僅能以甘特圖（Gource/Gantt）形式呈現微秒級的執行時序，更能透過流動事件（Flow Events）視覺化跨模組的因果關係，從而為開發者與架構師提供一個強大的除錯與效能優化工具。此方法論將有助於縮短複雜 WiFi 問題的平均修復時間（MTTR），並提升 Carrier-Grade CPE 軟體的品質與穩定性。

## 2. 緒論與背景架構分析

### 2.1 從 OpenWrt 到 prplOS 的演進

OpenWrt 長期以來作為嵌入式無線路由器的標準開源作業系統，其核心優勢在於強大的套件管理系統與高度可定制的構建環境。然而，隨著 ISP（網際網路服務供應商）對於遠端管理標準（如 USP/TR-369）及硬體抽象化的需求提升，原始的 OpenWrt 架構在跨平台兼容性與服務隔離性上顯得力不從心 1。prplOS 應運而生，它保留了 OpenWrt 的底層優勢（如 UCI 配置系統、ubus 匯流排），但在應用層引入了 Ambiorix 框架，旨在解決碎片化的匯流排介面問題。

Ambiorix 的核心哲學是「匯流排無關性」（Bus Agnostic）。在傳統 OpenWrt 中，應用程式直接依賴 libubus 進行通訊；而在 RDK-B 系統中則依賴 D-Bus 或 R-Bus。Ambiorix 定義了一套通用的 API（BAAPI），使得開發者撰寫的應用程式（外掛程式）可以在不修改程式碼的情況下，運行於 ubus、PCB 或其他私有匯流排之上 7。這種架構雖然極大地方便了軟體重用，但也使得執行路徑變得更加抽象與難以追蹤。

### 2.2 可觀測性挑戰：非同步與模組化

在 prplOS 的 WiFi 子系統中，一個簡單的操作（例如用戶更改 SSID）可能涉及以下流程：

1. **USP Agent** 接收來自雲端的控制器指令。
2. **Ambiorix Data Model** 層驗證參數並更新 TR-181 資料模型。
3. **libamxp Signal** 觸發事件通知（Event Notification）。
4. **WiFi Plugin (wld/prplMesh)** 的 FSM 接收信號，決定是否需要重啟無線電介面。
5. **FTA Layer** 呼叫底層晶片廠商提供的驅動程式 API。

在這個鏈條中，步驟 3 到 4 往往是透過非同步的信號機制完成的，這意味著我們無法單純透過堆疊追蹤（Stack Trace）來還原完整的呼叫鏈。此外，步驟 5 的硬體操作通常是阻塞式（Blocking）或耗時的，若缺乏精確的測量，很難區分是軟體邏輯延遲還是硬體回應緩慢。因此，構建一個能夠串聯這些異質層級的視覺化系統，成為了系統分析的關鍵。

## 3. Ambiorix 生態系統中的追蹤機制詳解

要實現架構視覺化，首先必須掌握 Ambiorix 框架提供的原生追蹤能力。Ambiorix 並非單一函式庫，而是一組協同工作的模組集合，其中 libamxp、libamxc 與 libsahtrace 在可觀測性中扮演了核心角色。

### 3.1 libamxp：事件驅動與信號槽機制

libamxp 是 Ambiorix 的核心模式庫，它實作了事件迴圈（Event Loop）與信號槽（Signal/Slot）模式，這是 prplOS 非同步通訊的基礎 9。

#### 3.1.1 信號管理器的運作原理

在 libamxp 中，任何物件都可以擁有一個信號管理器（Signal Manager）。當物件狀態改變時，它會透過 amxp\_sigmngr\_emit\_signal 函式發射信號。這個過程包含以下關鍵步驟：

1. **信號觸發**：應用程式呼叫發射函式，帶入信號名稱（如 dm:object-changed）與酬載數據（Variant）。
2. **槽查找**：信號管理器遍歷已註冊的槽（Slots）鏈結串列。
3. **同步執行**：在標準配置下，槽函式會被立即、同步地執行。這對於追蹤非常有利，因為這意味著信號發射與槽執行發生在同一個執行緒上下文中，我們可以利用執行緒 ID（TID）與時間戳記來關聯這兩者 11。

然而，由於 Ambiorix 支援延遲執行與佇列機制，某些情況下信號可能會被放入排程器中稍後執行。這也是為什麼單純依賴日誌順序可能不可靠，我們需要明確的「因果鏈」標記。

#### 3.1.2 視覺化意義

對於視覺化而言，libamxp 的信號發射點即是「流動事件」（Flow Event）的起點。如果我們能在信號發射時記錄下信號名稱與目標物件，並在槽函式開始執行時記錄下槽的名稱，我們就能在視覺化工具（如 Perfetto）中畫出一條連接兩者的箭頭，清晰地展示出「因為這個事件，導致了那個邏輯的執行」 12。

### 3.2 libsahtrace：追蹤區域（Trace Zones）的深度應用

libsahtrace 是 prplOS 的標準日誌與追蹤函式庫。與傳統的 syslog 相比，它引入了「追蹤區域」（Trace Zones）的概念，這對於在龐大的系統中過濾雜訊至關重要 4。

#### 3.2.1 區域化架構設計

每個應用程式或外掛程式可以定義多個追蹤區域。例如，wld 外掛程式可以定義：

* main：一般流程日誌。
* fsm：狀態機變遷日誌。
* fta：硬體呼叫日誌。
* events：內部事件處理日誌。

每個區域都有獨立的日誌等級（Level），通常使用整數表示（如 200 為 Info，500 為 Trace/Debug） 15。這種設計允許開發者在生產環境中僅開啟特定模組的高級別追蹤，而不影響整體系統效能。

#### 3.2.2 配置解析與 ODL 整合

libsahtrace 的強大之處在於它與 Ambiorix 的 ODL（Object Definition Language）配置系統深度整合。透過 libamxo 解析器，開發者可以在啟動時透過 ODL 檔案定義預設的追蹤行為 7。

程式碼片段

%config {  
 sahtrace = {  
 type = "syslog",  
 level = 200  
 };  
 trace-zones = {  
 "wld" = 500,  
 "fsm\_core" = 500,  
 "fta\_hal" = 500  
 };  
}

上述配置展示了如何利用 ODL 的 %config 區塊來初始化追蹤系統。trace-zones 是一個雜湊表（Hash Table），將區域名稱映射到整數等級。當 amxrt（Ambiorix Runtime）載入此 ODL 時，它會自動呼叫 libsahtrace 的 API 來設定這些區域。這意味著我們無需重新編譯程式碼，只需修改設定檔即可開啟針對 FSM 或 FTA 的深度視覺化數據源 7。

#### 3.2.3 執行時動態調整

除了靜態配置，prplOS 允許透過 ubus 或 CLI 工具在執行時動態調整追蹤區域。這對於診斷偶發性問題極為有用。例如，利用 XPON.set\_trace\_zone 或類似的模組特定指令，可以在不重啟服務的情況下，瞬間提升特定區域的詳細度，捕捉關鍵的故障現場數據 4。

### 3.3 libamxc：資料結構的序列化

在追蹤過程中，除了「發生了什麼」（事件），我們同樣關心「帶著什麼數據」（酬載）。Ambiorix 使用 libamxc 提供的 amxc\_var\_t 來封裝所有類型的數據 3。

為了實現視覺化，我們需要將這些 C 語言的結構體轉換為可被分析工具讀取的格式（如 JSON）。libamxc 內建了將 Variant 轉換為 JSON 字串的功能（amxc\_var\_dump 或 amxc\_var\_cast）。在我們的視覺化策略中，每當 FSM 發生狀態變遷或 FTA 被呼叫時，我們應當利用 libamxc 將相關的參數（如 SSID、Channel、MAC Address）序列化為 JSON，並作為 libsahtrace 的日誌內容輸出。這樣，後端的分析工具（如 Perfetto UI）就能夠在點擊特定事件時，展示出詳盡的參數資訊 18。

## 4. WiFi 子系統之 FSM 狀態機架構分析

在 prplOS 的 WiFi 架構中，邏輯決策的核心由有限狀態機（FSM）驅動。主要涉及的組件包括 wld（Wireless Daemon，負責傳統 WiFi 管理）與 prplMesh（基於 Beerocks，負責 EasyMesh 多存取點協同）。理解這些 FSM 的實作細節是進行視覺化的前提。

### 4.1 prplMesh (Beerocks) 的 C++ FSM 實作

prplMesh 採用現代 C++ 開發，其 FSM 實作通常封裝在類別中，利用列舉型別（Enum）定義狀態，並在主迴圈中透過 switch-case 結構處理事件 5。

#### 4.1.1 狀態定義與變遷

根據研究資料 5，BackhaulManager 的狀態機包含如 OPERATIONAL、WAIT\_FOR\_SCAN\_RESULTS、WIRELESS\_ASSOCIATE\_4ADDR 等狀態。每一個狀態代表了系統的一種穩定行為模式或等待階段。

關鍵的狀態變遷邏輯由 FSM\_MOVE\_STATE 巨集控制。這個巨集不僅僅是改變一個變數的值，它通常還包含了日誌記錄的功能 5。

C++

// 概念性代碼重建  
#**define** FSM\_MOVE\_STATE(new\_state) \  
 do { \  
 LOG(DEBUG) << "FSM Transition: " << m\_state << " --> " << new\_state; \  
 m\_state = new\_state; \  
 on\_state\_exit(m\_state); \  
 on\_state\_enter(new\_state); \  
 } while(0)

#### 4.1.2 視覺化截點策略

為了視覺化這個過程，我們需要攔截或解析 FSM\_MOVE\_STATE 產生的輸出。在 Perfetto 的模型中，狀態機的狀態應該被表示為「非同步切片」（Async Slice）。這意味著一個狀態有一個明確的「開始時間」與「結束時間」。

* **開始時間**：FSM\_MOVE\_STATE 被呼叫的時刻。
* **結束時間**：下一次 FSM\_MOVE\_STATE 被呼叫的時刻。

透過解析日誌中的 FSM\_MOVE\_STATE 條目，我們可以重建出時間軸上連續的狀態區塊。如果日誌中包含 FSM\_MOVE\_STATE(RESTART)，我們就在時間軸上結束當前的區塊，並開啟一個名為 "RESTART" 的新區塊。

### 4.2 WLD 的 C 語言 FSM 實作

與 prplMesh 不同，wld 以及部分傳統外掛程式可能使用 C 語言實作 FSM。這通常涉及函式指標陣列或大型的 switch 語句。雖然語法不同，但視覺化的原理一致：我們需要識別出改變狀態變數（如 fsm->current\_state）的程式碼位置，並確保該位置會觸發 libsahtrace 的日誌輸出。

### 4.3 巢狀狀態機與並行性

現代 WiFi 系統往往具有多層次的狀態機。例如，主 FSM 可能處於 OPERATIONAL 狀態，但其內部的子系統（如漫遊管理器）可能正在經歷 SCANNING -> CANDIDATE\_SELECTION -> HANDOVER 的過程。

在視覺化圖表中，這應該表現為「巢狀軌道」（Nested Tracks）。

* **Track 1 (Main FSM)**:
* **Track 2 (Roaming)**: [ Idle ][ Idle ]

利用 libsahtrace 的 Trace Zone，我們可以為不同的 FSM 分配不同的區域名稱（如 wld.fsm.main 與 wld.fsm.roam），從而在後處理階段將它們分離到不同的視覺化軌道上，清晰呈現並行邏輯的運作 18。

## 5. 功能表抽象層（FTA）之可覆寫機制與視覺化

如果說 FSM 是大腦，那麼 FTA（Function Table Abstraction）就是四肢。它是 prplOS 高層邏輯與底層晶片驅動程式（HAL）之間的介面。在視覺化分析中，FTA 層的重要性在於它揭示了「實際發生了什麼硬體操作」以及「這些操作耗時多久」。

### 5.1 FTA 結構體設計與 V-Table 模式

根據研究資料 6，FTA 本質上是一個包含多個函式指標的 C 語言結構體（struct）。這種設計實現了執行時的多型（Polymorphism），類似於 C++ 的虛擬函式表（V-Table）。

**表 1：FTA 結構體成員範例解析**

| **成員名稱 (Member Name)** | **類型 (Type)** | **功能描述 (Description)** | **典型用途 (Use Case)** |
| --- | --- | --- | --- |
| mfn\_wrad\_bgdfs\_enable | int (\*)(bool) | 啟用或停用背景 DFS 掃描 | 當 FSM 進入 DFS\_CAC 狀態時呼叫，以監聽雷達訊號。 |
| mfn\_wrad\_regdomain | int (\*)(char\*) | 設定無線電的監管區域碼 | 初始化階段或漫遊至不同國家代碼的網路時呼叫。 |
| mfn\_wrad\_supstd | int (\*)(int\*) | 獲取硬體支援的 WiFi 標準 | 確定是 802.11ac 還是 802.11ax，影響 FSM 的能力協商邏輯。 |
| mfn\_wrad\_getCurrentTxPow\_dBm | int (\*)(int\*) | 讀取當前發射功率 | 用於除錯訊號覆蓋範圍或功率控制演算法的驗證。 |

### 5.2 可覆寫特性與動態連結

「可覆寫」（Overridable）意味著這些函式指標在程式執行期間是可以被修改的。這通常發生在模組初始化階段：

1. 通用 wld 核心啟動，FTA 指標初始化為 NULL 或預設值。
2. 供應商特定的模組（如 mod-whm-mxl 或 mod-whm-qualcomm）載入。
3. 該模組將其內部的硬體操作函式位址賦值給全域的 FTA 結構體成員。

這種機制為我們提供了一個絕佳的「切入點」（Hook Point）。我們不需要修改封閉原始碼的驅動程式，也不需要重新編譯通用的 wld 核心，只需要在兩者之間插入一個「中介層」（Shim Layer）。

### 5.3 建構 FTA 視覺化探針（Instrumentation Shim）

為了視覺化 FTA 呼叫，我們可以利用 C 語言的指標特性，將 FTA 成員指向我們自定義的包裝函式（Wrapper Function）。

#### 5.3.1 實作邏輯

假設我們要追蹤 mfn\_wrad\_bgdfs\_enable：

C

// 定義原始函式指標類型  
typedef int (\*bgdfs\_func\_t)(bool);  
  
// 保存原始的硬體實作指標  
static bgdfs\_func\_t original\_bgdfs = NULL;  
  
// 定義我們的包裝函式  
int trace\_bgdfs\_enable(bool enable) {  
 // 1. 記錄開始事件 (Perfetto Phase 'B')  
 SAHTRACE\_Z("fta\_trace", 500, "{\"ph\":\"B\", \"name\":\"bgdfs\_enable\", \"args\":{\"enable\":%d}}", enable);  
   
 // 2. 呼叫原始硬體函式  
 int ret = original\_bgdfs(enable);  
   
 // 3. 記錄結束事件 (Perfetto Phase 'E')  
 SAHTRACE\_Z("fta\_trace", 500, "{\"ph\":\"E\", \"name\":\"bgdfs\_enable\", \"args\":{\"ret\":%d}}", ret);  
   
 return ret;  
}  
  
// 在初始化階段進行覆寫  
void install\_fta\_hooks(fta\_t\* fta\_struct) {  
 original\_bgdfs = fta\_struct->mfn\_wrad\_bgdfs\_enable;  
 fta\_struct->mfn\_wrad\_bgdfs\_enable = trace\_bgdfs\_enable;  
}

#### 5.3.2 阻塞式呼叫的視覺化價值

FTA 函式通常涉及與核心空間（Kernel Space）的 Netlink 通訊或 ioctl 呼叫，這些操作可能是阻塞的（Blocking）。如果在 Ambiorix 的主事件迴圈中呼叫了一個耗時 500ms 的 FTA 函式，整個應用程式將會凍結 500ms，導致信號處理延遲。

透過上述的包裝函式，我們可以在視覺化圖表中精確地看到每個 FTA 呼叫的「長度」。如果我們在 Perfetto 中看到一個 FTA 區塊異常的長，且對應的 FSM 狀態變遷被推遲，我們就立刻定位到了效能瓶頸的根源 5。

## 6. 整合式視覺化方法論：從日誌到圖表

掌握了資料源（Trace Zones, FSM, FTA）後，下一步是將這些分散的日誌數據整合為有意義的視覺化圖表。本節提出一套完整的資料處理管線。

### 6.1 資料攝取與標準化

由於 prplOS 的日誌可能混雜了系統訊息與其他雜訊，我們首先需要一個強大的解析器（Parser）。建議使用 Python 腳本進行後處理。

**日誌格式假設：**

[ZONE][LEVEL] MESSAGE

**解析策略：**

1. **Regex 提取**：使用正規表示式提取時間戳記、區域名稱與 JSON 酬載。
2. **上下文關聯**：由於日誌是線性的，但 FSM 是有狀態的，解析器需要維護每個 FSM 實例的「當前狀態」，以便在讀取到新狀態時，自動生成上一個狀態的「結束事件」。
3. **時鐘同步**：如果日誌包含來自不同設備（如 Mesh 中的多個節點）的數據，需要基於 NTP 時間進行校正，以確保因果關係的正確性。

### 6.2 映射至 Perfetto (Trace Event Format)

Perfetto 是 Google 開發的新一代系統追蹤工具，其基於 JSON 的 Trace Event 格式非常適合表現 prplOS 的架構 12。

#### 6.2.1 對應關係表

| **prplOS 概念** | **Perfetto 事件類型 (Phase)** | **參數映射 (Arguments)** | **視覺效果** |
| --- | --- | --- | --- |
| **Ambiorix Signal** | Instant Event (ph: 'i') | name: 信號名, scope: Process | 時間軸上的垂直標記，表示事件發生點。 |
| **Signal-Slot 關聯** | Flow Event (ph: 's', 't', 'f') | id: 唯一關聯 ID | 連接信號發射點與槽執行點的曲線箭頭，展示因果流。 |
| **FSM State** | Async Slice (ph: 'b', 'e') | cat: "fsm", name: 狀態名 | 跨越一段時間的長條，不同狀態可有不同顏色。 |
| **FTA Call** | Complete Event (ph: 'X') | dur: 執行時長, args: 參數 | 位於 FSM 下方的短條，展示硬體操作的實際佔用時間。 |

#### 6.2.2 實作細節：流動事件（Flow Events）

最關鍵的視覺化在於連接「軟體決策」與「硬體反應」。當 Ambiorix 發出信號時，我們在 JSON 中生成一個 flow\_start；當 FSM 處理該信號並呼叫 FTA 時，我們生成一個 flow\_end。這樣，Perfetto UI 會繪製一條線，直接告訴開發者：「因為這個 DM 變更，導致了這個硬體暫存器寫入」 13。

### 6.3 映射至 Mermaid (動態生成靜態圖)

除了時間軸視圖，我們有時需要驗證狀態機的邏輯正確性。透過解析日誌中的所有 FSM\_MOVE\_STATE 事件，我們可以生成 Mermaid 的 stateDiagram 24。

**生成腳本邏輯：**

遍歷日誌，對於每一對 (Previous\_State, New\_State)，生成一行 Mermaid 語法：

Previous\_State --> New\_State : Trigger\_Event

這將產出一張有向圖（Directed Graph），顯示系統在實際運行中走過的所有路徑。如果圖中出現了設計文件中未定義的變遷路徑，或者出現了死迴圈（Dead Loop），即可立即發現邏輯錯誤。

### 6.4 映射至 Gource (活動熱圖)

雖然 Gource 主要用於源代碼版控視覺化，但它也支援自定義日誌格式（Custom Log Format） 26。我們可以將 FSM 的狀態視為「檔案」，將執行緒視為「使用者」。

**格式：** timestamp|thread\_name|type|state\_path

例如：1678888888|MainThread|M|FSM/WLD/OPERATIONAL

這將產生一個動畫，展示系統在不同狀態之間的「跳躍」頻率。對於長時間運行的壓力測試，這種視覺化能直觀地揭示出系統最常停留在哪個狀態，或者哪個狀態切換最為頻繁（可能暗示振盪問題）。

## 7. 案例研究與綜合分析

### 7.1 案例一：DFS 雷達偵測與頻道切換

**情境**：5GHz 頻段的無線電偵測到雷達脈衝，必須立即切換頻道（DFS 事件）。

**視覺化分析流程**：

1. **起點**：Perfetto 時間軸上出現一個來自 Kernel 的 Netlink 事件，被 Ambiorix 轉換為信號 wifi:radar\_detected。
2. **傳遞**：一條 Flow 箭頭從該信號指向 wld 的 FSM 處理區塊。
3. **狀態變遷**：FSM 軌道顯示從 OPERATIONAL 瞬間變更為 DFS\_CAC\_WAIT（可用性檢查等待）。
4. **硬體操作**：緊接著狀態變更，FTA 軌道出現一個長條 mfn\_wrad\_bgdfs\_enable(false)，表示關閉背景掃描以規避干擾。
5. **異常診斷**：如果 FTA 長條的長度超過了法規要求的反應時間（例如 100ms），或者 FSM 在進入 DFS\_CAC\_WAIT 之前有不明顯的延遲，視覺化圖表將一目了然地暴露問題。

### 7.2 案例二：用戶端漫遊 (Roaming) 失敗

**情境**：用戶端設備嘗試從 AP-A 漫遊至 AP-B，但連線中斷。

**視覺化分析流程**：

1. **觸發**：AP-A 的 FSM 接收到 RSSI\_LOW 信號。
2. **決策**：FSM 進入 STEERING\_CANDIDATE\_SEARCH。
3. **互動**：FTA 軌道顯示密集的 mfn\_wrad\_get\_station\_stats 呼叫，這是系統在收集周邊訊號強度。
4. **斷裂點**：Perfetto 顯示 FSM 發出了 802.11v BTM Request（基本服務集轉換管理請求），但隨後長時間沒有收到 BTM Response。
5. **超時**：FSM 軌道顯示 TIMEOUT 狀態，隨後重置回 IDLE。
6. **洞察**：透過查看 FTA 呼叫的參數（透過 libamxc 序列化），我們發現發出的 BTM Request 中包含了錯誤的候選 AP 列表，導致用戶端拒絕漫遊。這是單純看狀態機變遷無法發現的數據層面錯誤。

## 8. 結論與未來展望

本研究展示了如何在 prplOS 的複雜架構中，利用 Ambiorix 的基礎設施構建一套深度的可觀測性解決方案。透過將 libsahtrace 的靈活配置、FSM 的邏輯狀態以及 FTA 的硬體操作進行有機結合，並映射到現代化的視覺化工具（Perfetto/Mermaid）上，我們成功地將「黑盒子」式的嵌入式開發轉變為透明的、可視化的工程流程。

這套方法論不僅適用於除錯，更可用於效能優化（找出 FTA 阻塞點）、自動化測試驗證（透過 Mermaid 比對設計規格）以及新進人員的架構培訓。未來，隨著 prplOS 引入更多微服務與容器化技術，這種跨層級、跨模組的分布式追蹤（Distributed Tracing）能力將成為 Carrier-Grade CPE 軟體開發的標準配備。透過進一步結合 eBPF 等核心層追蹤技術，我們將能實現從硬體中斷到雲端控制平面的全鏈路可觀測性，為下一代寬頻網路設備的穩定性奠定堅實基礎。

#### 引用的著作

1. prplOS - prpl Foundation, 檢索日期：2月 7, 2026， <https://prplfoundation.org/prplos/>
2. CPE Development with prplOS & prplMesh Expertise - Zilogic Systems, 檢索日期：2月 7, 2026， <https://zilogic.com/prpl-expertise.html>
3. prpl Foundation / components / ambiorix / libraries / libamxc - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/libraries/libamxc>
4. prpl Foundation / components / core / modules / mod-sahtrace - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/core/modules/mod-sahtrace>
5. prplMesh/agent/src/beerocks/slave/backhaul\_manager/backhaul\_manager\_thread.cpp at master - GitHub, 檢索日期：2月 7, 2026， <https://github.com/prplfoundation/prplMesh/blob/master/agent/src/beerocks/slave/backhaul_manager/backhaul_manager_thread.cpp>
6. [mxl-ospv2][updk 9.2] kernel crash when calling ... - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/mirrors/hostap/-/integrations/jira/issues/PCF-2250>
7. doc/odl.md · main · prpl Foundation / components / ambiorix / libraries / libamxo - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/libraries/libamxo/-/blob/main/doc/odl.md>
8. prpl Foundation / components / ambiorix / tutorials / getting-started - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/tutorials/getting-started>
9. include/amxp · main · prpl Foundation / components / ambiorix / libraries / libamxp - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/libraries/libamxp/-/tree/main/include/amxp?ref_type=heads>
10. prpl Foundation / components / ambiorix / libraries / libamxp - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/libraries/libamxp>
11. src/amxb\_ba\_wait\_for.c · v4.8.2 · prpl Foundation / components / ambiorix / libraries / libamxb - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/ambiorix/libraries/libamxb/-/blob/v4.8.2/src/amxb_ba_wait_for.c>
12. Visualizing external trace formats with Perfetto, 檢索日期：2月 7, 2026， <https://perfetto.dev/docs/getting-started/other-formats>
13. Android Jank detection with FrameTimeline - Perfetto Tracing Docs, 檢索日期：2月 7, 2026， <https://perfetto.dev/docs/data-sources/frametimeline>
14. prpl Foundation / components / core / libraries / libsahtrace - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/core/libraries/libsahtrace>
15. prpl Foundation / components / core / plugins / faultmonitor - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/core/plugins/faultmonitor>
16. prpl Foundation / components / core / modules / mod-sahtrace - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/core/modules/mod-sahtrace/-/tree/v0.0.10>
17. prpl Foundation / components / core / plugins / tr181-xpon - GitLab, 檢索日期：2月 7, 2026， <https://gitlab.com/prpl-foundation/components/core/plugins/tr181-xpon>
18. All my favorite tracing tools: eBPF, QEMU, Perfetto, new ones I built and more - Tristan Hume, 檢索日期：2月 7, 2026， <https://thume.ca/2023/12/02/tracing-methods/>
19. prplfoundation/prplMesh: This repository moved to https://gitlab.com/prpl-foundation/prplmesh/prplMesh - GitHub, 檢索日期：2月 7, 2026， <https://github.com/prplfoundation/prplMesh>
20. Flow events support · Issue #149 · wolfpld/tracy - GitHub, 檢索日期：2月 7, 2026， <https://github.com/wolfpld/tracy/issues/149>
21. lbirchler/strace-perfetto - GitHub, 檢索日期：2月 7, 2026， <https://github.com/lbirchler/strace-perfetto>
22. Converting from Perfetto to other trace formats, 檢索日期：2月 7, 2026， <https://perfetto.dev/docs/quickstart/traceconv>
23. omnitrace: Visualization and Features of Perfetto Traces (v1.0.0) - YouTube, 檢索日期：2月 7, 2026， <https://www.youtube.com/watch?v=7WN3N1hnCbI>
24. Usage - Mermaid Chart, 檢索日期：2月 7, 2026， <https://mermaid.ai/open-source/config/usage.html>
25. javascript - How to render a mermaid flowchart dynamically? - Stack Overflow, 檢索日期：2月 7, 2026， <https://stackoverflow.com/questions/65212332/how-to-render-a-mermaid-flowchart-dynamically>
26. Repository visualization with Gource - Fab Academy, 檢索日期：2月 7, 2026， <https://fabacademy.org/2015/doc/gource.html>
27. Visualizing our 7 years of online dialogue with Gource - Software (SSNA) - Edgeryders, 檢索日期：2月 7, 2026， <https://edgeryders.eu/t/visualizing-our-7-years-of-online-dialogue-with-gource/11905>