Asset Servant 正確定義

作者: Manus AI
日期: 2025年10月14日
版本: 2.0 (修正版)

核心定義

Asset Servant 是什麼?

Asset Servant 是連結 3D Model Transformation 與時序資料庫即時 IOT Tag 資訊,並儲存具有 Asset 狀態和 Behavior 的虛擬分身物件。

Asset Servant 的三大核心職責

1. 連結層 (Linking Layer)

連結 3D Model Transformation 與即時 IOT Tag 資訊

```
3D Model (USD) in Omniverse
    ↕ (雙向連結)
Asset Servant (虛擬分身物件)
    ↕ (雙向連結)
時序資料庫 (即時 IOT Tag 資訊)
```

具體功能: - 將 3D 模型的 Transformation (位置、旋轉、縮放) 與實際設備狀態同步 - 將即時 IOT Tag 資訊反映到 3D 模型的視覺化表現 - 實現 3D 視覺化與實際設備的即時聯動

2. 狀態容器 (State Container)

儲存 Asset 的即時狀態

Asset Servant 在運行時儲存: - ✅ 當前運行狀態 (running, stopped, alarm, maintenance) - ✅ 即時 IOT Tag 值 (溫度、壓力、流量等) - ✅ 計算屬性 (效率、健康度等) - ✅ 告警狀態 - ✅ 3D Model Transformation 狀態

3. Behavior 執行引擎 (Behavior Engine)

執行 Asset 的行為邏輯

Asset Servant 可以執行: - ✅ 狀態機邏輯 (State Machine) - ✅ 告警規則 (Alarm Rules) - ✅ 自動化腳本 (Automation Scripts) - ✅ 事件響應 (Event Handlers)

完整架構圖


Omniverse (3D 視覺化層)
USD 3D Model: PUMP-001
- Transformation (位置、旋轉)
- Visual Properties (顏色、可見性)
- Animations (動畫狀態)
↕ (雙向同步)
Asset Servant (虛擬分身物件) ⭐
Asset ID: PUMP-001
【狀態容器】
- current_state: "running"
- discharge_pressure: 5.2 bar (即時值)
- flow_rate: 120 m³/h (即時值)
- efficiency: 87% (計算值)
- health_score: 92 (計算值)
- alarm_status: "normal"
- model_transform: {x, y, z, rotation}
【Tag 映射表】
- discharge_pressure → TDengine.pump001_pressure
- flow_rate → PI.PLANT1_PUMP001_FLOW
【Behavior 邏輯】
- on_pressure_high(): trigger_alarm()
- on_state_change(): update_3d_color()
- calculate_efficiency(): ...
↕ (查詢/訂閱)
時序資料庫 (即時 IOT Tag 資訊)
TDengine / PI System / InfluxDB
- pump001_pressure: 5.2 bar (2025-10-14 10:30:15)
- pump001_flow: 120 m³/h (2025-10-14 10:30:15)

Asset Servant 的完整實現

核心類別定義

```python
from typing import Dict, Any, Callable, Optional
from datetime import datetime
from enum import Enum

class AssetState(Enum):
    """資產狀態枚舉"""
    STOPPED = "stopped"
    STARTING = "starting"
    RUNNING = "running"
    STOPPING = "stopping"
    ALARM = "alarm"
    MAINTENANCE = "maintenance"
    ERROR = "error"

class AssetServant:
    """
    Asset Servant - 虛擬分身物件

    三大核心職責:
    1. 連結 3D Model Transformation 與即時 IOT Tag 資訊
    2. 儲存 Asset 的即時狀態
    3. 執行 Asset 的 Behavior 邏輯
    """

    def __init__(self, asset_id: str, iadl_definition: dict):
        self.asset_id = asset_id
        self.iadl_definition = iadl_definition

        # ===== 1. 狀態容器 =====
        self.current_state: AssetState = AssetState.STOPPED
        self.telemetry: Dict[str, Any] = {}  # 即時 IOT Tag 值
        self.properties: Dict[str, Any] = {}  # 靜態屬性
        self.computed_values: Dict[str, Any] = {}  # 計算值
        self.alarm_status: Dict[str, str] = {}  # 告警狀態

        # 3D Model Transformation 狀態
        self.model_transform = {
            'position': {'x': 0, 'y': 0, 'z': 0},
            'rotation': {'x': 0, 'y': 0, 'z': 0},
            'scale': {'x': 1, 'y': 1, 'z': 1}
        }
        self.visual_properties = {
            'color': '#00FF00',  # 綠色 = 正常
            'opacity': 1.0,
            'visible': True
        }

        # ===== 2. Tag 映射表 =====
        self.tag_mappings = self._build_tag_mappings(iadl_definition)

        # ===== 3. Behavior 邏輯 =====
        self.behaviors: Dict[str, Callable] = {}
        self.event_handlers: Dict[str, list] = {}
        self._register_behaviors()

        # 時序資料庫適配器
        self.tsdb_adapters = {}
```

```
        # Omniverse USD 連接
        self.usd_stage = None
        self.usd_prim_path = None

    # ========================================
    # 1. 連結層功能
    # ========================================

    async def sync_from_iot(self):
        """
        從時序資料庫同步即時 IOT Tag 資訊到 Asset Servant

        流程:
        1. 查詢時序資料庫獲取最新值
        2. 更新 self.telemetry
        3. 觸發 Behavior 邏輯
        4. 同步到 3D Model
        """
        for tag_id, mapping in self.tag_mappings.items():
            # 從時序資料庫獲取最新值
            latest_value = await self._query_latest_value(tag_id, mapping)

            # 更新即時值
            old_value = self.telemetry.get(tag_id)
            self.telemetry[tag_id] = latest_value

            # 觸發值變化事件
            if old_value != latest_value:
                await self._on_telemetry_changed(tag_id, old_value,
latest_value)

    async def sync_to_3d_model(self):
        """
        將 Asset Servant 的狀態同步到 Omniverse 3D Model

        流程:
        1. 更新 USD Transformation
        2. 更新 Visual Properties (顏色、可見性)
        3. 觸發動畫
        """
        if not self.usd_stage or not self.usd_prim_path:
            return

        # 獲取 USD Prim
        prim = self.usd_stage.GetPrimAtPath(self.usd_prim_path)

        # 更新 Transformation
        from pxr import UsdGeom, Gf
        xform = UsdGeom.Xformable(prim)

        translate_op = xform.AddTranslateOp()
        translate_op.Set(Gf.Vec3d(
            self.model_transform['position']['x'],
            self.model_transform['position']['y'],
            self.model_transform['position']['z']
        ))

        # 更新顏色 (根據狀態)
        color = self._get_state_color()
        # 設置 USD 材質顏色...

    async def sync_from_3d_model(self):
        """
        從 Omniverse 3D Model 同步 Transformation 到 Asset Servant
```

```
        使用場景:
        - 用戶在 Omniverse 中手動移動了 3D 模型
        - 需要將新位置同步回 Asset Servant
        """
        if not self.usd_stage or not self.usd_prim_path:
            return

        prim = self.usd_stage.GetPrimAtPath(self.usd_prim_path)
        from pxr import UsdGeom
        xform = UsdGeom.Xformable(prim)

        # 讀取 Transformation
        translate_op = xform.GetOrderedXformOps()[0]
        position = translate_op.Get()

        self.model_transform['position'] = {
            'x': position[0],
            'y': position[1],
            'z': position[2]
        }

    async def _query_latest_value(self, tag_id: str, mapping: dict) -> Any:
        """從時序資料庫查詢最新值"""
        db_type = mapping['type']
        adapter = self.tsdb_adapters.get(db_type)

        if db_type == 'tdengine':
            result = await adapter.query(
                measurement=mapping['measurement'],
                limit=1,
                order='desc'
            )
            return result[0]['value'] if result else None

        elif db_type == 'pi_point':
            result = await adapter.query(
                measurement=mapping['point_name'],
                limit=1
            )
            return result[0]['value'] if result else None

    # ========================================
    # 2. 狀態容器功能
    # ========================================

    def get_telemetry(self, tag_id: str) -> Any:
        """獲取即時 IOT Tag 值"""
        return self.telemetry.get(tag_id)

    def get_all_telemetry(self) -> Dict[str, Any]:
        """獲取所有即時值"""
        return self.telemetry.copy()

    def get_state(self) -> AssetState:
        """獲取當前狀態"""
        return self.current_state

    async def set_state(self, new_state: AssetState):
        """
        設置狀態 (會觸發狀態變化事件)
        """
        old_state = self.current_state
        self.current_state = new_state

        # 觸發狀態變化事件
        await self._on_state_changed(old_state, new_state)
```

```python
        # 更新 3D 模型顏色
        await self.sync_to_3d_model()

    def compute_efficiency(self) -> float:
        """
        計算效率 (範例 Behavior)

        根據即時 IOT Tag 值計算資產效率
        """
        # 從 IADL 定義中獲取額定值
        rated_flow = self.properties.get('rated_flow', 150)  # m³/h
        rated_power = self.properties.get('rated_power', 15)  # kW

        # 獲取即時值
        actual_flow = self.telemetry.get('flow_rate', 0)
        actual_power = self.telemetry.get('power_consumption', 0)

        # 計算效率
        if actual_power > 0:
            efficiency = (actual_flow / rated_flow) / (actual_power /
rated_power) * 100
            self.computed_values['efficiency'] = round(efficiency, 2)
            return efficiency
        return 0.0

    def compute_health_score(self) -> float:
        """
        計算健康度評分 (範例 Behavior)
        """
        score = 100.0

        # 根據振動值扣分
        vibration = self.telemetry.get('vibration', 0)
        if vibration > 10:
            score -= (vibration - 10) * 2

        # 根據溫度扣分
        temperature = self.telemetry.get('bearing_temperature', 0)
        if temperature > 80:
            score -= (temperature - 80) * 1.5

        self.computed_values['health_score'] = max(0, round(score, 2))
        return score

    # ========================================
    # 3. Behavior 執行引擎
    # ========================================

    def  register behaviors(self):
        """註冊 Behavior 邏輯"""
        # 從 IADL 定義中讀取 Behaviors
        behaviors = self.iadl_definition.get('behaviors', [])

        for behavior in behaviors:
            behavior_type = behavior.get('type')

            if behavior type == 'alarm rule':
                self._register_alarm_rule(behavior)
            elif behavior_type == 'state_machine':
                self. register state machine(behavior)
            elif behavior_type == 'automation':
                self._register_automation(behavior)

    def _register_alarm_rule(self, rule: dict):
        """
```

```
        註冊告警規則

        範例:
        {
          "type": "alarm_rule",
          "name": "high_pressure_alarm",
          "condition": "discharge_pressure > 6.0",
          "severity": "high",
          "message": "出口壓力過高"
        }
        """
        rule_name = rule['name']
        condition = rule['condition']

        async def check_alarm():
            # 評估條件表達式
            if self._evaluate_condition(condition):
                await self._trigger_alarm(rule_name, rule)

        self.behaviors[rule_name] = check_alarm

    async def _on_telemetry_changed(self, tag_id: str, old_value: Any,
new_value: Any):
        """
        IOT Tag 值變化事件處理

        當時序資料庫的值更新時觸發
        """
        # 執行所有相關的 Behavior
        for behavior_name, behavior_func in self.behaviors.items():
            await behavior_func()

        # 重新計算計算值
        self.compute_efficiency()
        self.compute_health_score()

        # 同步到 3D 模型
        await self.sync_to_3d_model()

        # 觸發自定義事件處理器
        if tag_id in self.event_handlers:
            for handler in self.event_handlers[tag_id]:
                await handler(old_value, new_value)

    async def _on_state_changed(self, old_state: AssetState, new_state:
AssetState):
        """
        狀態變化事件處理
        """
        print(f"Asset {self.asset_id} state changed: {old_state} ->
{new_state}")

        # 更新 3D 模型顏色
        self.visual_properties['color'] = self._get_state_color()

        # 記錄到時序資料庫
        await self._log_state_change(old_state, new_state)

    def _get_state_color(self) -> str:
        """根據狀態返回顏色"""
        color_map = {
            AssetState.STOPPED: '#808080',      # 灰色
            AssetState.STARTING: '#FFFF00',     # 黃色
            AssetState.RUNNING: '#00FF00',      # 綠色
            AssetState.STOPPING: '#FFA500',     # 橙色
            AssetState.ALARM: '#FF0000',        # 紅色
```

```
            AssetState.MAINTENANCE: '#0000FF',  # 藍色
            AssetState.ERROR: '#8B0000'         # 深紅色
        }
        return color_map.get(self.current_state, '#FFFFFF')

    async def _trigger_alarm(self, alarm_name: str, rule: dict):
        """觸發告警"""
        self.alarm_status[alarm_name] = {
            'severity': rule['severity'],
            'message': rule['message'],
            'timestamp': datetime.now().isoformat(),
            'acknowledged': False
        }

        # 如果是高嚴重性告警,切換到告警狀態
        if rule['severity'] == 'high':
            await self.set_state(AssetState.ALARM)

    def _evaluate_condition(self, condition: str) -> bool:
        """
        評估條件表達式

        範例: "discharge_pressure > 6.0"
        """
        # 簡單實現:使用 eval (生產環境應使用安全的表達式解析器)
        context = self.telemetry.copy()
        try:
            return eval(condition, {"__builtins__": {}}, context)
        except:
            return False

    # ========================================
    # 輔助方法
    # ========================================

    def _build_tag_mappings(self, iadl: dict) -> dict:
        """建立 Tag 映射表"""
        mappings = {}
        for tag in iadl.get('data_tags', []):
            mappings[tag['tag_id']] = tag.get('source', {})
        return mappings

    async def _log_state_change(self, old_state: AssetState, new_state:
AssetState):
        """記錄狀態變化到時序資料庫"""
        # 寫入狀態變化事件
        pass

    # ========================================
    # 公開 API
    # ========================================

    async def start(self):
        """啟動 Asset Servant"""
        # 連接到 Omniverse
        await self._connect_to_omniverse()

        # 啟動即時數據同步
        await self._start_telemetry_sync()

        # 設置狀態為運行
        await self.set_state(AssetState.RUNNING)

    async def stop(self):
        """停止 Asset Servant"""
        await self.set_state(AssetState.STOPPED)
```

```
        # 停止即時數據同步
        await self._stop_telemetry_sync()

    async def _connect_to_omniverse(self):
        """連接到 Omniverse USD Stage"""
        # 從 IADL 獲取 USD 模型路徑
        model_file = self.iadl_definition.get('geometry', {}).get('model_file')

        if model_file:
            from pxr import Usd
            self.usd_stage = Usd.Stage.Open(model_file)
            self.usd_prim_path = f"/World/Assets/{self.asset_id}"

    async def _start_telemetry_sync(self):
        """啟動即時數據同步 (定期從時序資料庫拉取)"""
        import asyncio

        async def sync_loop():
            while self.current_state == AssetState.RUNNING:
                await self.sync_from_iot()
                await asyncio.sleep(1)  # 每秒同步一次

        asyncio.create_task(sync_loop())

    async def _stop_telemetry_sync(self):
        """停止即時數據同步"""
        pass
```

IADL 中的 Behaviors 定義

```yaml
# pump001.iadl.yaml

asset:
  asset_id: PUMP-001
  asset_type: centrifugal_pump

  # 3D 模型
  geometry:
    model_file: "omniverse://server/projects/plant1/models/pump001.usd"

  # 數據標籤
  data_tags:
    - tag_id: discharge_pressure
      source:
        type: tdengine
        measurement: pump001_pressure

    - tag_id: flow_rate
      source:
        type: pi_point
        point_name: PLANT1_PUMP001_FLOW

    - tag_id: vibration
      source:
        type: tdengine
        measurement: pump001_vibration

  # 靜態屬性
  properties:
    rated_flow: 150  # m³/h
    rated_power: 15  # kW

  # Behaviors 定義 (Asset Servant 會執行)
  behaviors:
    # 告警規則
    - type: alarm_rule
      name: high_pressure_alarm
      condition: "discharge_pressure > 6.0"
      severity: high
      message: "出口壓力過高"

    - type: alarm_rule
      name: high_vibration_alarm
      condition: "vibration > 15"
      severity: medium
      message: "振動異常"

    # 狀態機
    - type: state_machine
      states:
        - stopped
        - starting
        - running
        - stopping
        - alarm
      transitions:
        - from: stopped
          to: starting
          condition: "start_command == true"
        - from: starting
          to: running
```

```
          condition: "flow_rate > 10"
        - from: running
          to: alarm
          condition: "discharge_pressure > 6.0"

    # 自動化腳本
    - type: automation
      name: auto_shutdown_on_high_temp
      trigger: "bearing_temperature > 90"
      actions:
        - stop_pump()
        - send_notification("泵浦因高溫自動停機")
```

Asset Servant 的儲存內容

Asset Servant 儲存 (運行時記憶體 + 可選持久化)

```
{
    # 運行時狀態 (記憶體)
    "current_state": "running",
    "telemetry": {
        "discharge_pressure": 5.2,  # 即時值
        "flow_rate": 120,
        "vibration": 8.5
    },
    "computed_values": {
        "efficiency": 87.5,
        "health_score": 92
    },
    "alarm_status": {
        "high_pressure_alarm": {
            "active": false
        }
    },
    "model_transform": {
        "position": {"x": 10, "y": 0, "z": 5},
        "rotation": {"x": 0, "y": 90, "z": 0}
    },
    "visual_properties": {
        "color": "#00FF00",
        "opacity": 1.0
    },

    # Tag 映射表 (記憶體,從 IADL 解析)
    "tag_mappings": {
        "discharge_pressure": {
            "type": "tdengine",
            "measurement": "pump001_pressure"
        }
    },

    # Behavior 邏輯 (記憶體,從 IADL 解析)
    "behaviors": {
        "high_pressure_alarm": <function>,
        "compute_efficiency": <function>
    }
}
```

Asset Servant 不儲存

- ❌ USD 3D Model 檔案本身 (在 Omniverse Nucleus)
- ❌ 歷史時序數據 (在 TDengine/PI)
- ❌ IADL 定義檔案 (在 Git/檔案系統)

完整的數據流

場景: 壓力感測器值更新

```
1. 物理泵浦壓力變化
   ↓
2. PLC 採集壓力值
   ↓
3. 寫入 TDengine
   pump001_pressure = 5.8 bar
   ↓
4. Asset Servant 定期同步 (每秒)
   await sync_from_iot()
   ↓
5. Asset Servant 更新內部狀態
   self.telemetry['discharge_pressure'] = 5.8
   ↓
6. 觸發 Behavior 邏輯
   - 檢查告警規則: 5.8 < 6.0 (正常)
   - 重新計算效率
   ↓
7. 同步到 3D Model
   await sync_to_3d_model()
   - 更新 USD Prim 的顏色 (綠色 = 正常)
   ↓
8. Omniverse 中的 3D 模型即時更新
   用戶看到泵浦顏色為綠色
```

場景: 壓力過高觸發告警

```
1. TDengine 中壓力值 = 6.5 bar
   ↓
2. Asset Servant 同步
   self.telemetry['discharge_pressure'] = 6.5
   ↓
3. 觸發告警規則
   condition: "discharge_pressure > 6.0" = True
   ↓
4. Asset Servant 執行告警 Behavior
   - 設置告警狀態
   - 切換到 ALARM 狀態
   ↓
5. 狀態變化觸發視覺更新
   self.visual_properties['color'] = '#FF0000' (紅色)
   ↓
6. 同步到 3D Model
   USD Prim 顏色變為紅色
   ↓
7. Omniverse 中泵浦變紅,用戶立即察覺異常
```

總結

Asset Servant 的正確定位

Asset Servant 是一個虛擬分身物件,具有三大核心職責:

1. **連結層**:
2. 連結 3D Model Transformation 與即時 IOT Tag 資訊
3. 實現 3D 視覺化與實際設備的雙向同步
4. **狀態容器**:
5. 儲存 Asset 的即時狀態 (運行狀態、IOT Tag 值、計算值)
6. 儲存 3D Model Transformation 狀態
7. 儲存告警狀態
8. **Behavior 執行引擎**:
9. 執行告警規則
10. 執行狀態機邏輯
11. 執行自動化腳本

12. 計算衍生值 (效率、健康度)

與其他組件的關係

```
 IADL 定義 (藍圖)
     ↓ (解析)
Asset Servant (虛擬分身物件,運行時)
     ↕ (雙向同步)
3D Model (視覺化) + 時序資料庫 (即時數據)
```

Asset Servant 是 IDTF 數位分身架構的**核心執行時組件**,它將靜態的 IADL 定義轉化為動態的、有狀態的、有行為的虛擬分身物件。