IDTF 技術白皮書 (第二版)

Industrial Digital Twins Framework - Technical Whitepaper

版本: 2.0
日期: 2025 年 1 月
作者: IDTF 技術團隊

第三部分:整合方案

8. 企業層整合 (ERP/MES)

8.1 ERP 整合

8.1.1 支援的 ERP 系統

- ✅ SAP ERP (ECC/S4HANA)
- ✅ Oracle ERP Cloud
- ✅ Microsoft Dynamics 365
- ✅ Infor CloudSuite
- ✅ QAD
- ✅ 鼎新 ERP
- ✅ 正航 ERP

8.1.2 資料流

從 ERP 到 IDTF: - 生產計劃 (Production Plan) - 物料需求 (Material Requirement) - 採購訂單 (Purchase Order) - 銷售訂單 (Sales Order)

從 IDTF 到 ERP: - 實際生產進度 - 實際物料消耗 - 實際設備運行時間 - 實際能源消耗 - 預測性維護需求

8.1.3 SAP ERP Connector

```python
class SAPERPConnector:
    def __init__(self, host, sysnr, client, user, password):
        self.conn = Connection(
            ashost=host,
            sysnr=sysnr,
            client=client,
            user=user,
            passwd=password
        )

    def get_production_orders(self, plant, date_from, date_to):
        # 呼叫 SAP RFC: BAPI_PRODORD_GET_LIST
        result = self.conn.call(
            'BAPI_PRODORD_GET_LIST',
            PLANT=plant,
            DATE_FROM=date_from,
            DATE_TO=date_to
        )

        # 發布到 Kafka
        for order in result['ORDERS']:
            event = {
                "source": "SAP ERP",
                "type": "production_order",
                "order_number": order['ORDERID'],
                "material": order['MATERIAL'],
                "quantity": order['TARGET_QTY'],
                "start_date": order['BASIC_START_DATE'],
                "end_date": order['BASIC_END_DATE'],
                "timestamp": datetime.now()
            }
            kafka_producer.send("erp.production_plan", event)

    def update_production_progress(self, order_number, quantity_produced):
        # 呼叫 SAP RFC: BAPI_PRODORDCONF_CREATE_TT
        result = self.conn.call(
            'BAPI_PRODORDCONF_CREATE_TT',
            ORDERID=order_number,
            YIELD=quantity_produced
        )

        return result
```

8.2 MES 整合

8.2.1 支援的 MES 系統

- ✅ SAP MES (ME/MII)
- ✅ Siemens Opcenter
- ✅ Rockwell FactoryTalk
- ✅ Dassault DELMIA
- ✅ Honeywell Uniformance

- ✅ Wonderware MES
- ✅ 自建 MES

8.2.2 資料流

從 MES 到 IDTF: - 工單資訊 (Work Order) - 品質資料 (SPC) - 物料追蹤 (Traceability) - OEE 資料 - 生產排程

從 IDTF 到 MES: - 設備狀態更新 (即時) - OEE 計算結果 - AI 預測和建議 - 異常警報

8.2.3 Siemens Opcenter Connector

```python
class OpCenterConnector:
    def __init__(self, base_url, api_key):
        self.base_url = base_url
        self.api_key = api_key

    def get_work_orders(self, status):
        # 呼叫 Opcenter REST API
        response = requests.get(
            f"{self.base_url}/api/workorders",
            headers={"Authorization": f"Bearer {self.api_key}"},
            params={"status": status}
        )

        work_orders = response.json()

        # 發布到 Kafka
        for wo in work_orders:
            event = {
                "source": "Siemens Opcenter",
                "type": "work_order",
                "wo_id": wo['id'],
                "product": wo['product'],
                "quantity": wo['quantity'],
                "start_time": wo['start_time'],
                "status": wo['status'],
                "timestamp": datetime.now()
            }
            kafka_producer.send("mes.work_order", event)

    def update_equipment_status(self, equipment_id, status):
        # 更新設備狀態到 Opcenter
        response = requests.put(
            f"{self.base_url}/api/equipment/{equipment_id}/status",
            headers={"Authorization": f"Bearer {self.api_key}"},
            json={"status": status}
        )

        return response.json()
```

8.3 應用案例

案例 1: 訂單交付預測

傳統方式: 1-2 小時,資訊可能已過時
IDTF 方式: 5 分鐘,資訊即時準確

```
class OrderDeliveryPredictor:
    def predict_delivery(self, order_id):
        # 從 ERP 獲取訂單資訊
        order = erp_connector.get_order(order_id)

        # 從 MES 獲取工單進度
        work_orders = mes_connector.get_work_orders_by_order(order_id)

        # 從 IDTF 獲取設備即時狀態
        equipment_status = []
        for wo in work_orders:
            status = ndh_api.get_equipment_status(wo['equipment_id'])
            equipment_status.append(status)

        # AI 預測完成時間
        predicted_completion = ai_model.predict_completion_time(
            work_orders,
            equipment_status
        )

        # 計算準時交付機率
        on_time_probability = ai_model.predict_on_time_probability(
            predicted_completion,
            order['due_date']
        )

        return {
            "order_id": order_id,
            "current_progress": sum([wo['progress'] for wo in work_orders]) /
len(work_orders),
            "predicted_completion": predicted_completion,
            "due_date": order['due_date'],
            "on_time_probability": on_time_probability
        }
```

案例 2: 設備故障影響分析

```
class EquipmentFailureImpactAnalyzer:
    def analyze_impact(self, equipment_id):
        # 從 IDTF 獲取設備故障資訊
        equipment = ndh_api.get_equipment(equipment_id)

        # 從 MES 獲取受影響的工單
        affected_work_orders =
mes_connector.get_work_orders_by_equipment(equipment_id)

        # 從 ERP 獲取受影響的訂單
        affected_orders = []
        for wo in affected_work_orders:
            order = erp_connector.get_order_by_work_order(wo['wo_id'])
            affected_orders.append(order)

        # 計算影響
        total_delay_hours = self.estimate_repair_time(equipment_id)
        total_affected_orders = len(affected_orders)
        total_revenue_at_risk = sum([order['value'] for order in
affected_orders])

        # 自動重新排程
        new_schedule = mes_connector.reschedule(affected_work_orders,
total_delay_hours)

        # 更新 ERP 交付日期
        for order in affected_orders:
            new_delivery_date = order['due_date'] +
timedelta(hours=total_delay_hours)
            erp_connector.update_delivery_date(order['id'], new_delivery_date)

        # 通知客戶
        for order in affected_orders:
            self.notify_customer(order, new_delivery_date)

        return {
            "equipment_id": equipment_id,
            "estimated_repair_time": total_delay_hours,
            "affected_orders": total_affected_orders,
            "revenue_at_risk": total_revenue_at_risk,
            "new_schedule": new_schedule
        }
```

---

9. 控制層整合 (SCADA/PLC)

9.1 SCADA 整合

9.1.1 支援的 SCADA 系統

- ✅ Wonderware System Platform / InTouch
- ✅ Ignition by Inductive Automation

- ✅ Siemens WinCC
- ✅ Rockwell FactoryTalk View
- ✅ GE iFIX
- ✅ Schneider Citect SCADA

9.1.2 Wonderware 雙向整合

匯出 (IADL → Wonderware):

```
class IADLToWonderwareExporter:
    def export_to_csv(self, iadl_assets, output_file):
        # 生成 Wonderware CSV 格式
        rows = []
        rows.append(["TagName", "DataType", "Address", "Description",
"EngUnits", "AlarmHiHi", "AlarmHi", "AlarmLo", "AlarmLoLo"])

        for asset in iadl_assets:
            for tag_name, tag_def in asset['tags']['scada'].items():
                row = [
                    f"{asset['id']}.{tag_name}",
                    tag_def['data_type'],
                    tag_def.get('address', ''),
                    f"{asset['name']} - {tag_name}",
                    tag_def.get('unit', ''),
                    tag_def.get('alarm_hihi', ''),
                    tag_def.get('alarm_hi', ''),
                    tag_def.get('alarm_lo', ''),
                    tag_def.get('alarm_lolo', '')
                ]
                rows.append(row)

        # 寫入 CSV
        with open(output_file, 'w', newline='', encoding='utf-8-sig') as f:
            writer = csv.writer(f)
            writer.writerows(rows)
```

匯入 (Wonderware → IADL):

```python
class WonderwareToIADLImporter:
    def import_from_csv(self, csv_file):
        # 讀取 Wonderware CSV
        with open(csv_file, 'r', encoding='utf-8-sig') as f:
            reader = csv.DictReader(f)
            tags = list(reader)

        # 轉換為 IADL 格式
        assets = {}
        for tag in tags:
            # 解析 TagName (格式: AssetID.TagName)
            parts = tag['TagName'].split('.')
            asset_id = parts[0]
            tag_name = '.'.join(parts[1:])

            if asset_id not in assets:
                assets[asset_id] = {
                    "id": asset_id,
                    "tags": {"scada": {}}
                }

            assets[asset_id]['tags']['scada'][tag_name] = {
                "data_type": tag['DataType'],
                "address": tag.get('Address', ''),
                "unit": tag.get('EngUnits', ''),
                "alarm_hihi": tag.get('AlarmHiHi', ''),
                "alarm_hi": tag.get('AlarmHi', ''),
                "alarm_lo": tag.get('AlarmLo', ''),
                "alarm_lolo": tag.get('AlarmLoLo', '')
            }

        return list(assets.values())
```

自動同步:

```python
class WonderwareIADLSync:
    def __init__(self, wonderware_server, iadl_repo):
        self.wonderware = wonderware_server
        self.iadl_repo = iadl_repo

    def sync(self):
        # 檢測 Wonderware 變更
        wonderware_tags = self.wonderware.get_all_tags()
        wonderware_hash = self.calculate_hash(wonderware_tags)

        # 檢測 IADL 變更
        iadl_assets = self.iadl_repo.get_all_assets()
        iadl_hash = self.calculate_hash(iadl_assets)

        # 比較並同步
        if wonderware_hash != self.last_wonderware_hash:
            # Wonderware 有變更,更新 IADL
            self.update_iadl_from_wonderware(wonderware_tags)

        if iadl_hash != self.last_iadl_hash:
            # IADL 有變更,更新 Wonderware
            self.update_wonderware_from_iadl(iadl_assets)

        # 更新 hash
        self.last_wonderware_hash = wonderware_hash
        self.last_iadl_hash = iadl_hash
```

9.2 PLC 整合

9.2.1 支援的 PLC 系統

- ✅ Siemens S7-300/400/1200/1500
- ✅ Allen-Bradley ControlLogix/CompactLogix
- ✅ Schneider Modicon M340/M580
- ✅ Mitsubishi iQ-R/iQ-F
- ✅ Omron NJ/NX

9.2.2 Siemens S7 整合

```python
class SiemensS7Connector:
    def __init__(self, ip, rack, slot):
        self.client = snap7.client.Client()
        self.client.connect(ip, rack, slot)

    def read_db(self, db_number, start, size):
        # 讀取 DB
        data = self.client.db_read(db_number, start, size)
        return data

    def write_db(self, db_number, start, data):
        # 寫入 DB
        self.client.db_write(db_number, start, data)

    def subscribe_tags(self, tag_list):
        # 輪詢標籤
        while True:
            for tag in tag_list:
                value = self.read_db(
                    tag['db_number'],
                    tag['start'],
                    tag['size']
                )

                # 發布到 Kafka
                event = {
                    "source": "Siemens S7",
                    "equipment_id": tag['equipment_id'],
                    "tag_name": tag['tag_name'],
                    "value": value,
                    "timestamp": datetime.now()
                }
                kafka_producer.send("plc.tag_change", event)

            time.sleep(0.1)  # 100ms 輪詢週期
```

9.2.3 Allen-Bradley 整合

```
class AllenBradleyConnector:
    def __init__(self, ip):
        self.comm = PLC()
        self.comm.IPAddress = ip

    def read_tag(self, tag_name):
        # 讀取標籤
        ret = self.comm.Read(tag_name)
        return ret.Value

    def write_tag(self, tag_name, value):
        # 寫入標籤
        ret = self.comm.Write(tag_name, value)
        return ret.Status == "Success"

    def subscribe_tags(self, tag_list):
        # 輪詢標籤
        while True:
            for tag in tag_list:
                value = self.read_tag(tag['tag_name'])

                # 發布到 Kafka
                event = {
                    "source": "Allen-Bradley",
                    "equipment_id": tag['equipment_id'],
                    "tag_name": tag['tag_name'],
                    "value": value,
                    "timestamp": datetime.now()
                }
                kafka_producer.send("plc.tag_change", event)

            time.sleep(0.1)
```

---

10. 設計層整合 (CAD/PLM)

10.1 CAD 整合

10.1.1 支援的 CAD 系統

- ✅ AutoCAD Plant 3D
- ✅ AVEVA E3D
- ✅ Bentley OpenPlant
- ✅ Intergraph SmartPlant 3D

10.1.2 AutoCAD Plant 3D 整合

```
// AutoCAD Plant 3D Plugin (C#)
public class IADLExtractor
{
    public List<Asset> ExtractAssets()
    {
        var assets = new List<Asset>();

        // 獲取所有設備
        var equipment = PlantApplication.CurrentProject.ProjectParts
            .OfType<Equipment>();

        foreach (var equip in equipment)
        {
            var asset = new Asset
            {
                Id = equip.Tag,
                Name = equip.Description,
                Type = equip.PartSizeProperties.NominalDiameter,
                Manufacturer = equip.Manufacturer,
                Model = equip.Model,
                Location = new Location
                {
                    Plant = PlantApplication.CurrentProject.ProjectName,
                    Position = new Position
                    {
                        X = equip.Position.X,
                        Y = equip.Position.Y,
                        Z = equip.Position.Z
                    }
                }
            };

            assets.Add(asset);
        }

        return assets;
    }

    public void ExportToIADL(List<Asset> assets, string outputFile)
    {
        // 序列化為 YAML
        var serializer = new SerializerBuilder().Build();
        var yaml = serializer.Serialize(assets);

        // 寫入檔案
        File.WriteAllText(outputFile, yaml);
    }
}
```

10.2 PLM 整合

10.2.1 支援的 PLM 系統

- ✅ Siemens Teamcenter
- ✅ PTC Windchill

- ✅ Dassault ENOVIA (3DEXPERIENCE)
- ✅ SAP PLM

10.2.2 Siemens Teamcenter 整合

```python
class TeamcenterConnector:
    def __init__(self, base_url, username, password):
        self.base_url = base_url
        self.session = self.login(username, password)

    def login(self, username, password):
        # 登入 Teamcenter
        response = requests.post(
            f"{self.base_url}/tc/rest/login",
            json={"username": username, "password": password}
        )
        return response.json()['session_id']

    def get_item(self, item_id):
        # 獲取 Item
        response = requests.get(
            f"{self.base_url}/tc/rest/items/{item_id}",
            headers={"Authorization": f"Bearer {self.session}"}
        )
        return response.json()

    def get_bom(self, item_id):
        # 獲取 BOM
        response = requests.get(
            f"{self.base_url}/tc/rest/items/{item_id}/bom",
            headers={"Authorization": f"Bearer {self.session}"}
        )
        return response.json()

    def sync_to_iadl(self, item_id):
        # 同步到 IADL
        item = self.get_item(item_id)
        bom = self.get_bom(item_id)

        # 轉換為 IADL 格式
        asset = {
            "id": item['item_id'],
            "name": item['object_name'],
            "type": item['object_type'],
            "manufacturer": item.get('manufacturer', ''),
            "model": item.get('model', ''),
            "parameters": item.get('properties', {}),
            "bom": bom
        }

        # 儲存到 IADL Repository
        iadl_repo.save_asset(asset)

        return asset
```

---

11. AI/ML 整合

11.1 AI/ML 架構

11.2 預測性維護

```python
class PredictiveMaintenanceModel:
    def __init__(self):
        self.model = self.load_model()

    def load_model(self):
        # 載入預訓練模型
        return joblib.load('predictive_maintenance_model.pkl')

    def predict_failure(self, equipment_id, lookback_hours=24):
        # 從 NDH 獲取歷史資料
        end_time = datetime.now()
        start_time = end_time - timedelta(hours=lookback_hours)

        data = ndh_api.get_equipment_history(
            equipment_id,
            start_time,
            end_time
        )

        # 特徵工程
        features = self.extract_features(data)

        # 預測
        prediction = self.model.predict(features)
        probability = self.model.predict_proba(features)[0][1]

        # 計算剩餘使用壽命 (RUL)
        rul = self.calculate_rul(features)

        return {
            "equipment_id": equipment_id,
            "failure_predicted": bool(prediction[0]),
            "failure_probability": probability,
            "remaining_useful_life_hours": rul,
            "recommended_action": self.get_recommendation(probability, rul)
        }

    def extract_features(self, data):
        # 提取特徵
        features = {
            "temperature_mean": np.mean(data['temperature']),
            "temperature_std": np.std(data['temperature']),
            "temperature_max": np.max(data['temperature']),
            "vibration_mean": np.mean(data['vibration']),
            "vibration_std": np.std(data['vibration']),
            "vibration_max": np.max(data['vibration']),
            "runtime_hours": np.sum(data['status'] == 'Running') / 60,
            # ... 更多特徵
        }
        return pd.DataFrame([features])

    def calculate_rul(self, features):
        # 計算剩餘使用壽命
        # 使用迴歸模型預測
        rul_model = joblib.load('rul_model.pkl')
        rul = rul_model.predict(features)[0]
        return max(0, rul)

    def get_recommendation(self, probability, rul):
        if probability > 0.8 or rul < 24:
            return "立即安排維護"
        elif probability > 0.5 or rul < 72:
```

```
            return "24小時內安排維護"
        elif probability > 0.3 or rul < 168:
            return "一週內安排維護"
        else:
            return "正常監控"
```

11.3 品質異常檢測

```python
class QualityAnomalyDetector:
    def __init__(self):
        self.model = IsolationForest(contamination=0.01)
        self.is_trained = False

    def train(self, equipment_id, days=30):
        # 獲取歷史正常資料
        end_time = datetime.now()
        start_time = end_time - timedelta(days=days)

        data = ndh_api.get_equipment_history(
            equipment_id,
            start_time,
            end_time
        )

        # 特徵工程
        features = self.extract_features(data)

        # 訓練模型
        self.model.fit(features)
        self.is_trained = True

    def detect_anomaly(self, equipment_id):
        if not self.is_trained:
            raise Exception("Model not trained")

        # 獲取即時資料
        data = ndh_api.get_equipment_realtime(equipment_id)

        # 特徵工程
        features = self.extract_features([data])

        # 預測
        prediction = self.model.predict(features)
        anomaly_score = self.model.score_samples(features)

        is_anomaly = prediction[0] == -1

        if is_anomaly:
            # 分析異常原因
            root_cause = self.analyze_root_cause(data, features)

            # 發送警報
            self.send_alert(equipment_id, root_cause, anomaly_score[0])

        return {
            "equipment_id": equipment_id,
            "is_anomaly": is_anomaly,
            "anomaly_score": anomaly_score[0],
            "root_cause": root_cause if is_anomaly else None
        }

    def extract_features(self, data):
        # 提取特徵
        features = []
        for d in data:
            feature = {
                "temperature": d['temperature'],
                "pressure": d['pressure'],
                "vibration": d['vibration'],
                "speed": d['speed'],
```

```
                # ... 更多特徵
            }
            features.append(feature)
        return pd.DataFrame(features)

    def analyze_root_cause(self, data, features):
        # 分析異常原因
        # 使用 SHAP 或其他可解釋性方法
        root_causes = []

        if data['temperature'] > 200:
            root_causes.append("溫度過高")
        if data['vibration'] > 10:
            root_causes.append("振動過大")
        if data['pressure'] < 40:
            root_causes.append("壓力過低")

        return ", ".join(root_causes)
```

11.4 製程優化

```python
class ProcessOptimizer:
    def __init__(self):
        self.model = None

    def train(self, equipment_id, days=90):
        # 獲取歷史資料
        end_time = datetime.now()
        start_time = end_time - timedelta(days=days)

        data = ndh_api.get_equipment_history(
            equipment_id,
            start_time,
            end_time
        )

        # 準備訓練資料
        X = data[['temperature', 'pressure', 'speed']]  # 輸入參數
        y = data['quality_score']  # 輸出品質

        # 訓練模型
        self.model = RandomForestRegressor(n_estimators=100)
        self.model.fit(X, y)

    def optimize_parameters(self, equipment_id, constraints):
        # 定義優化目標
        def objective(params):
            temperature, pressure, speed = params

            # 預測品質
            quality = self.model.predict([[temperature, pressure, speed]])[0]

            # 目標: 最大化品質
            return -quality

        # 定義約束
        bounds = [
            (constraints['temperature_min'], constraints['temperature_max']),
            (constraints['pressure_min'], constraints['pressure_max']),
            (constraints['speed_min'], constraints['speed_max'])
        ]

        # 優化
        result = minimize(
            objective,
            x0=[180, 50, 100],  # 初始值
            bounds=bounds,
            method='L-BFGS-B'
        )

        optimal_params = {
            "temperature": result.x[0],
            "pressure": result.x[1],
            "speed": result.x[2],
            "predicted_quality": -result.fun
        }

        return optimal_params
```
