FDL v0.1 規範文件

專案名稱: Factory Description Language (FDL) v0.1
版本: 0.1
作者: Michael Lin 林志錚
組織: HTFA/Digital Twins Alliance
日期: 2025-10-22
狀態: Draft（草案）

執行摘要

本文件定義 Factory Description Language (FDL) v0.1 的完整規範，包括語法、語義、約束和驗證規則。FDL 是用於描述工廠佈局、資產實例、連接關係和約束的聲明式語言。

核心目標： - 提供明確的 FDL 語法和語義定義 - 定義 FDL → USD 組合策略 - 建立驗證規則和約束 - 支援 NDH 解析和 Asset Instance 生成

關鍵決策： - 使用 YAML 作為主要格式，JSON 作為備選 - 採用 USD Reference + Variant 策略處理資產實例 - 明確定義座標系統和單位制 - 支援碰撞邊界（AABB）和安全距離約束

目錄

---

1. FDL 概述

1.1 什麼是 FDL?

Factory Description Language (FDL) 是一種聲明式語言，用於描述工廠的物理佈局、資產實例、連接關係和約束。

核心概念：

- **Site（廠區）**：工廠的頂層容器
- **Area（區域）**：廠區內的功能區域（如泵房、控制室）
- **Asset Instance（資產實例）**：引用 IADL 定義的資產類型，並指定其位置和變換
- **Connection（連接）**：資產之間的物理連接（如管線、電纜）
- **Constraint（約束）**：安全距離、碰撞邊界等約束條件

1.2 FDL 的作用

1. **工廠設計**：在 FDL Designer 中設計工廠佈局
2. **資產實例化**：從 IADL 資產類型創建具體實例
3. **碰撞檢測**：基於 AABB 和安全距離進行碰撞檢測
4. **NDH 解析**：NDH 讀取 FDL 並生成 Asset Instance 和 Tag Instance
5. **USD 場景生成**：將 FDL 轉換為 USD 場景用於 3D 視覺化

1.3 FDL 與 IADL 的關係

```
 IADL（資產類型定義）
    ↓ 引用
FDL（工廠佈局 + 資產實例）
    ↓ 解析
NDH（Asset Instance + Tag Instance）
    ↓ 同步
Omniverse USD（3D 場景）
```

2. FDL v0.1 Schema 定義

2.1 頂層結構

```yaml
# FDL v0.1 頂層結構
fdl_version: "0.1"  # FDL 版本號（必填）

# 全域座標系統與單位制（必填）
units:
  length: "mm"      # 長度單位：mm, cm, m, in, ft
  angle: "deg"      # 角度單位：deg, rad
  up_axis: "Z"      # 向上軸：Y, Z
  handedness: "right"  # 手性：right, left

# 廠區資訊（必填）
site:
  name: "DemoPlantA"  # 廠區名稱
  site_id: "site_001"  # 廠區 ID（可選）
  location:  # 地理位置（可選）
    latitude: 24.7736
    longitude: 120.9436
    altitude: 50.0

  # 建築物列表（可選）
  buildings: []

  # 區域列表（必填）
  areas: []

  # 公用系統（可選）
  utilities: []

  # 全域約束（可選）
  global_constraints: {}
```

2.2 Area（區域）結構

```yaml
# Area 結構
areas:
  - name: "PumpRoom01"  # 區域名稱（必填）
    area_id: "area_001"  # 區域 ID（可選）
    type: "production"  # 區域類型：production, storage, control, utility

    # 區域邊界（可選）
    boundary:
      type: "box"  # box, polygon
      dimensions:  # 對於 box 類型
        width: 10000  # mm
        depth: 8000   # mm
        height: 5000  # mm
      origin:  # 區域原點（相對於 Site）
        x: 0
        y: 0
        z: 0

    # 資產實例列表（必填）
    instances: []

    # 連接列表（可選）
    connections: []

    # 區域約束（可選）
    constraints: {}
```

2.3 Asset Instance（資產實例）結構

```yaml
# Asset Instance 結構
instances:
  - instance_id: "PR01_Pump_A"  # 實例 ID（必填，全域唯一）
    ref_asset: "Pump_001"  # 引用的 IADL 資產 ID（必填）
    name: "Pump A"  # 實例名稱（可選）

    # Transform（變換）（必填）
    transform:
      translation:  # 平移（相對於 Area 原點）
        x: 0
        y: 0
        z: 0
      rotation:  # 旋轉（歐拉角，單位由 units.angle 指定）
        x: 0
        y: 0
        z: 0
      scale:  # 縮放
        x: 1.0
        y: 1.0
        z: 1.0

    # Tag 覆寫（可選）
    tags_overrides: []

    # 碰撞邊界（可選，如果未指定則使用 IADL 中的定義）
    collision_bounds:
      type: "aabb"  # aabb（軸對齊包圍盒）, obb（定向包圍盒）, sphere
      min: [-500, -500, 0]  # AABB 最小點（相對於實例原點）
      max: [500, 500, 2000]  # AABB 最大點

    # 約束（可選）
    constraints:
      clearance_mm: 300  # 安全距離（毫米）
      access_required: true  # 是否需要維護通道
      fixed: false  # 是否固定（不可移動）

    # 元數據（可選）
    metadata:
      manufacturer: "Grundfos"
      model: "CR 64-2"
      serial_number: "SN123456"
      installation_date: "2024-01-15"
```

2.4 Connection（連接）結構

```yaml
# Connection 結構
connections:
  - connection_id: "conn_001"  # 連接 ID（必填）
    type: "pipe"  # 連接類型：pipe, cable, duct
    name: "Inlet Pipe"  # 連接名稱（可選）

    # 起點和終點
    from:
      instance_id: "PR01_Pump_A"  # 起點實例 ID
      port: "inlet"  # 起點端口（可選）
    to:
      instance_id: "PR01_Tank_A"  # 終點實例 ID
      port: "outlet"  # 終點端口（可選）

    # 路徑（可選）
    path:
      type: "polyline"  # polyline, bezier
      points:  # 路徑點列表（相對於 Area 原點）
        - [0, 0, 1000]
        - [1000, 0, 1000]
        - [1000, 1000, 1000]

    # 連接屬性（可選）
    properties:
      diameter_mm: 100  # 管徑（毫米）
      material: "stainless_steel"  # 材料
      pressure_rating: "PN16"  # 壓力等級

    # 約束（可選）
    constraints:
      min_bend_radius_mm: 300  # 最小彎曲半徑
      clearance_mm: 100  # 安全距離
```

2.5 Tag Override（Tag 覆寫）結構

```yaml
# Tag Override 結構
tags_overrides:
  - tag_id: "temp_sensor_01"  # Tag ID（引用 IADL 中的 Tag）
    instance_tag_id: "PR01_Pump_A_Temp"  # 實例 Tag ID（可選）

    # 覆寫 Tag 的局部變換（可選）
    local_transform:
      translation: [0.5, 0.0, 1.2]
      rotation: [0.0, 0.0, 0.0]
      scale: [1.0, 1.0, 1.0]

    # 覆寫 Tag 的屬性（可選）
    properties:
      alarm_high: 85.0
      alarm_low: 10.0

    # 映射到外部系統（可選）
    mappings:
      scada_tag: "PLC1.DB10.DBD0"
      historian_tag: "Plant.PumpRoom.PumpA.Temperature"
```

2.6 Utility（公用系統）結構

```yaml
# Utility 結構
utilities:
  - utility_id: "util_001"  # 公用系統 ID（必填）
    type: "electrical"  # 類型：electrical, water, gas, compressed_air, hvac
    name: "Main Power Distribution"  # 名稱（可選）

    # 公用系統屬性（可選）
    properties:
      voltage: "380V"
      frequency: "50Hz"
      capacity: "500kW"

    # 連接到的資產實例（可選）
    connected_instances:
      - instance_id: "PR01_Pump_A"
        connection_point: "power_inlet"
      - instance_id: "PR01_Pump_B"
        connection_point: "power_inlet"
```

2.7 Constraint（約束）結構

```yaml
# Global Constraints（全域約束）
global_constraints:
  default_clearance_mm: 300  # 預設安全距離
  min_aisle_width_mm: 1200  # 最小通道寬度
  max_stack_height_mm: 5000  # 最大堆疊高度

  # 碰撞檢測設定
  collision_detection:
    enabled: true
    check_aabb: true  # 檢查 AABB 碰撞
    check_clearance: true  # 檢查安全距離
    ignore_vertical_clearance: false  # 是否忽略垂直方向的安全距離

  # 佈局規則
  layout_rules:
    - rule_id: "rule_001"
      type: "min_distance"
      description: "Pumps must be at least 500mm apart"
      applies_to:
        asset_types: ["Pump"]
      parameters:
        min_distance_mm: 500
```

---

3. FDL → USD 組合策略

3.1 USD 組合架構

決策：使用 **USD Reference + Variant** 策略

理由： - **USD Reference**：實現資產的重用和實例化，節省記憶體 - **Variant**：處理 LOD（細節層次）和型號差異 - **Instancing**：對於大量相同資產，使用 USD Instancing 提升效能

3.2 USD 場景結構

```
/World (Xform)
    ├── /Metadata (Custom data)
    │   ├── fdl_version: "0.1"
    │   ├── units: {...}
    │   └── site: {...}
    │
    ├── /Site_DemoPlantA (Xform)
    │   ├── /Area_PumpRoom01 (Xform)
    │   │   ├── /Instance_PR01_Pump_A (Xform)
    │   │   │   ├── references: asset://Pump_001/pump.usd
    │   │   │   ├── variantSets: ["LOD", "Model"]
    │   │   │   ├── xformOp:translate
    │   │   │   ├── xformOp:rotateXYZ
    │   │   │   └── xformOp:scale
    │   │   │
    │   │   ├── /Instance_PR01_Pump_B (Xform)
    │   │   │   └── ...
    │   │   │
    │   │   └── /Connections (Scope)
    │   │       ├── /Conn_001 (BasisCurves)
    │   │       └── ...
    │   │
    │   └── /Area_ControlRoom01 (Xform)
    │       └── ...
    │
    └── /Utilities (Scope)
        ├── /Util_Electrical_001
        └── ...
```

3.3 USD Reference 實作

```python
# FDL → USD 組合實作
from pxr import Usd, UsdGeom, Sdf, Gf

class FDLToUSDComposer:
    """FDL 到 USD 的組合器"""

    def __init__(self, fdl_data: dict, asset_library_path: str):
        self.fdl_data = fdl_data
        self.asset_library_path = asset_library_path

    def compose(self, output_usd_path: str) -> bool:
        """
        將 FDL 組合為 USD 場景

        Args:
            output_usd_path: 輸出 USD 檔案路徑

        Returns:
            success: 是否成功
        """
        # 創建 USD Stage
        stage = Usd.Stage.CreateNew(output_usd_path)

        # 設定 Stage Metadata
        self._setup_stage_metadata(stage)

        # 創建 World 根節點
        world_prim = UsdGeom.Xform.Define(stage, '/World')

        # 創建 Site
        site_prim = self._create_site(stage, self.fdl_data['site'])

        # 創建 Areas 和 Asset Instances
        for area_data in self.fdl_data['site']['areas']:
            self._create_area(stage, site_prim, area_data)

        # 創建 Utilities
        if 'utilities' in self.fdl_data['site']:
            self._create_utilities(stage, self.fdl_data['site']['utilities'])

        # 儲存 Stage
        stage.Save()

        print(f"[FDL→USD] Successfully composed USD scene: {output_usd_path}")
        return True

    def _setup_stage_metadata(self, stage: Usd.Stage):
        """設定 Stage Metadata"""
        units = self.fdl_data['units']

        # 設定 Up Axis
        up_axis_token = UsdGeom.Tokens.z if units['up_axis'] == 'Z' else
UsdGeom.Tokens.y
        UsdGeom.SetStageUpAxis(stage, up_axis_token)

        # 設定 Meters Per Unit
        length_unit = units['length']
        meters_per_unit_map = {
            'mm': 0.001,
            'cm': 0.01,
            'm': 1.0,
            'in': 0.0254,
            'ft': 0.3048
        }
        UsdGeom.SetStageMetersPerUnit(stage, meters_per_unit_map[length_unit])

        # 設定自定義 Metadata
        stage.SetMetadata('customLayerData', {
            'fdl_version': self.fdl_data['fdl_version'],
            'units': units,
            'site_name': self.fdl_data['site']['name']
        })

    def _create_site(self, stage: Usd.Stage, site_data: dict) -> Usd.Prim:
        """創建 Site Prim"""
        site_name = site_data['name']
        site_path = f'/World/Site_{site_name}'

        site_prim = UsdGeom.Xform.Define(stage, site_path).GetPrim()

        # 設定 Site 屬性
        if 'location' in site_data:
            site_prim.SetCustomDataByKey('location', site_data['location'])

        return site_prim

    def _create_area(self, stage: Usd.Stage, site_prim: Usd.Prim, area_data:
dict):
        """創建 Area Prim 和其中的 Asset Instances"""
        area_name = area_data['name']
        area_path = f"{site_prim.GetPath()}/Area_{area_name}"

        area_prim = UsdGeom.Xform.Define(stage, area_path).GetPrim()

        # 設定 Area 邊界（如果有）
        if 'boundary' in area_data:
            self._create_area_boundary(stage, area_prim, area_data['boundary'])

        # 創建 Asset Instances
        for instance_data in area_data['instances']:
            self._create_asset_instance(stage, area_prim, instance_data)

        # 創建 Connections
        if 'connections' in area_data:
            self._create_connections(stage, area_prim,
area_data['connections'])

    def _create_asset_instance(self, stage: Usd.Stage, area_prim: Usd.Prim,
                               instance_data: dict):
        """創建 Asset Instance（使用 USD Reference）"""
        instance_id = instance_data['instance_id']
        ref_asset = instance_data['ref_asset']

        # 創建 Instance Prim
        instance_path = f"{area_prim.GetPath()}/Instance_{instance_id}"
        instance_prim = UsdGeom.Xform.Define(stage, instance_path).GetPrim()

        # 添加 USD Reference
        asset_usd_path = f"
{self.asset_library_path}/{ref_asset}/{ref_asset}.usd"
        instance_prim.GetReferences().AddReference(asset_usd_path)

        # 設定 Transform
        xformable = UsdGeom.Xformable(instance_prim)
        transform_data = instance_data['transform']

        # Translation
        translation = Gf.Vec3d(
            transform_data['translation']['x'],
            transform_data['translation']['y'],
            transform_data['translation']['z']
        )
        xformable.AddTranslateOp().Set(translation)

        # Rotation (歐拉角)
        rotation = Gf.Vec3d(
            transform_data['rotation']['x'],
            transform_data['rotation']['y'],
            transform_data['rotation']['z']
        )
        xformable.AddRotateXYZOp().Set(rotation)

        # Scale
        scale = Gf.Vec3d(
            transform_data['scale']['x'],
            transform_data['scale']['y'],
            transform_data['scale']['z']
        )
        xformable.AddScaleOp().Set(scale)

        # 設定 Variant (如果有)
        if 'variant' in instance_data:
            self._set_variant(instance_prim, instance_data['variant'])

        # 設定碰撞邊界 (如果有)
        if 'collision_bounds' in instance_data:
            instance_prim.SetCustomDataByKey('collision_bounds',
                                            instance_data['collision_bounds'])

        # 設定約束 (如果有)
        if 'constraints' in instance_data:
            instance_prim.SetCustomDataByKey('constraints',
                                            instance_data['constraints'])

        # 設定元數據 (如果有)
        if 'metadata' in instance_data:
            instance_prim.SetCustomDataByKey('metadata',
                                            instance_data['metadata'])

    def _set_variant(self, prim: Usd.Prim, variant_data: dict):
        """設定 Variant"""
        for variant_set_name, variant_name in variant_data.items():
            variant_set = prim.GetVariantSets().AddVariantSet(variant_set_name)
            variant_set.SetVariantSelection(variant_name)

    def _create_connections(self, stage: Usd.Stage, area_prim: Usd.Prim,
                           connections_data: list):
        """創建 Connections (使用 BasisCurves)"""
        connections_scope = stage.DefinePrim(f"
{area_prim.GetPath()}/Connections", 'Scope')

        for conn_data in connections_data:
            conn_id = conn_data['connection_id']
            conn_path = f"{connections_scope.GetPath()}/Conn_{conn_id}"

            # 創建 BasisCurves
            curves = UsdGeom.BasisCurves.Define(stage, conn_path)

            # 設定路徑點
            if 'path' in conn_data and 'points' in conn_data['path']:
                points = [Gf.Vec3f(*pt) for pt in conn_data['path']['points']]
                curves.GetPointsAttr().Set(points)

                # 設定曲線類型
                curves.GetTypeAttr().Set(UsdGeom.Tokens.linear)
                curves.GetWrapAttr().Set(UsdGeom.Tokens.nonperiodic)

                # 設定頂點數量
                curves.GetCurveVertexCountsAttr().Set([len(points)])

            # 設定連接屬性
            if 'properties' in conn_data:
                curves.GetPrim().SetCustomDataByKey('properties',
conn_data['properties'])

    def _create_area_boundary(self, stage: Usd.Stage, area_prim: Usd.Prim,
                              boundary_data: dict):
        """創建 Area 邊界（使用 Cube 或 Mesh）"""
        if boundary_data['type'] == 'box':
            boundary_path = f"{area_prim.GetPath()}/Boundary"
            cube = UsdGeom.Cube.Define(stage, boundary_path)

            dimensions = boundary_data['dimensions']
            # USD Cube 的 size 是邊長，需要轉換
            # 這裡簡化處理，實際應該使用 Mesh
            cube.GetSizeAttr().Set(max(dimensions['width'],
dimensions['depth'], dimensions['height']))

            # 設定位置
            if 'origin' in boundary_data:
                origin = boundary_data['origin']
                xformable = UsdGeom.Xformable(cube)
                xformable.AddTranslateOp().Set(Gf.Vec3d(origin['x'],
origin['y'], origin['z']))

    def _create_utilities(self, stage: Usd.Stage, utilities_data: list):
        """創建 Utilities"""
        utilities_scope = stage.DefinePrim('/World/Utilities', 'Scope')

        for util_data in utilities_data:
            util_id = util_data['utility_id']
            util_path = f"{utilities_scope.GetPath()}/Util_{util_id}"

            util_prim = stage.DefinePrim(util_path, 'Scope')

            # 設定屬性
            if 'properties' in util_data:
                util_prim.SetCustomDataByKey('properties',
util_data['properties'])

            # 設定連接的實例
            if 'connected_instances' in util_data:
                util_prim.SetCustomDataByKey('connected_instances',
                                             util_data['connected_instances'])
```

3.4 USD Variant 使用

```python
# 在 IADL Asset USD 檔案中定義 Variant
# asset://Pump_001/pump.usd

from pxr import Usd, UsdGeom

# 創建 Asset Stage
stage = Usd.Stage.CreateNew('pump.usd')

# 創建根 Prim
root_prim = UsdGeom.Xform.Define(stage, '/Pump_001').GetPrim()

# 創建 LOD Variant Set
lod_variant_set = root_prim.GetVariantSets().AddVariantSet('LOD')

# LOD0 (高精度)
lod_variant_set.AddVariant('LOD0')
lod_variant_set.SetVariantSelection('LOD0')
with lod_variant_set.GetVariantEditContext():
    # 在這裡定義 LOD0 的幾何
    mesh = UsdGeom.Mesh.Define(stage, '/Pump_001/Geometry_LOD0')
    # ... 設定高精度幾何

# LOD1 (中精度)
lod_variant_set.AddVariant('LOD1')
lod_variant_set.SetVariantSelection('LOD1')
with lod_variant_set.GetVariantEditContext():
    # 在這裡定義 LOD1 的幾何
    mesh = UsdGeom.Mesh.Define(stage, '/Pump_001/Geometry_LOD1')
    # ... 設定中精度幾何

# LOD2 (低精度)
lod_variant_set.AddVariant('LOD2')
lod_variant_set.SetVariantSelection('LOD2')
with lod_variant_set.GetVariantEditContext():
    # 在這裡定義 LOD2 的幾何
    cube = UsdGeom.Cube.Define(stage, '/Pump_001/Geometry_LOD2')
    # ... 設定低精度幾何 (簡化為立方體)

# 創建 Model Variant Set (用於不同型號)
model_variant_set = root_prim.GetVariantSets().AddVariantSet('Model')

# Model A
model_variant_set.AddVariant('ModelA')
# Model B
model_variant_set.AddVariant('ModelB')

# 預設選擇
lod_variant_set.SetVariantSelection('LOD1')
model_variant_set.SetVariantSelection('ModelA')

stage.Save()
```

3.5 USD Instancing 優化

對於大量相同資產，使用 USD Instancing：

```
# 啟用 USD Instancing
instance_prim.SetInstanceable(True)
```

效果： - 記憶體使用大幅減少 - 渲染效能提升 - 適用於大型工廠場景（如數百個相同的感測器）

---

4. 驗證規則與約束

4.1 Schema 驗證

使用 JSON Schema 驗證 FDL 檔案：

```python
# fdl_validator.py
import jsonschema
import yaml
from typing import Dict, List, Tuple

class FDLValidator:
    """FDL 驗證器"""

    def __init__(self, schema_path: str):
        """
        初始化驗證器

        Args:
            schema_path: JSON Schema 檔案路徑
        """
        with open(schema_path, 'r') as f:
            self.schema = yaml.safe_load(f)

    def validate(self, fdl_data: dict) -> Tuple[bool, List[str]]:
        """
        驗證 FDL 數據

        Args:
            fdl_data: FDL 數據字典

        Returns:
            (is_valid, errors): 是否有效和錯誤列表
        """
        errors = []

        try:
            # Schema 驗證
            jsonschema.validate(instance=fdl_data, schema=self.schema)
        except jsonschema.ValidationError as e:
            errors.append(f"Schema validation error: {e.message}")
            return False, errors

        # 語義驗證
        semantic_errors = self._validate_semantics(fdl_data)
        errors.extend(semantic_errors)

        # 約束驗證
        constraint_errors = self._validate_constraints(fdl_data)
        errors.extend(constraint_errors)

        is_valid = len(errors) == 0
        return is_valid, errors

    def _validate_semantics(self, fdl_data: dict) -> List[str]:
        """語義驗證"""
        errors = []

        # 1. 檢查 instance_id 唯一性
        instance_ids = set()
        for area in fdl_data['site']['areas']:
            for instance in area['instances']:
                instance_id = instance['instance_id']
                if instance_id in instance_ids:
                    errors.append(f"Duplicate instance_id: {instance_id}")
                instance_ids.add(instance_id)

        # 2. 檢查 ref_asset 引用有效性
        # （這裡需要訪問 Asset Library 來驗證）

        # 3. 檢查 Connection 引用的 instance_id 存在
        for area in fdl_data['site']['areas']:
            if 'connections' in area:
                for conn in area['connections']:
                    from_id = conn['from']['instance_id']
                    to_id = conn['to']['instance_id']

                    if from_id not in instance_ids:
                        errors.append(f"Connection {conn['connection_id']}: "
                                    f"from instance_id '{from_id}' not found")
                    if to_id not in instance_ids:
                        errors.append(f"Connection {conn['connection_id']}: "
                                    f"to instance_id '{to_id}' not found")

        return errors

    def _validate_constraints(self, fdl_data: dict) -> List[str]:
        """約束驗證"""
        errors = []

        # 1. 檢查安全距離
        # 2. 檢查碰撞
        # 3. 檢查佈局規則

        # （這裡需要實作具體的約束檢查邏輯）

        return errors
```

4.2 碰撞檢測

```
# collision_detector.py
from typing import List, Tuple
import numpy as np

class AABBCollisionDetector:
    """AABB 碰撞檢測器"""

    def detect_collisions(self, instances: List[dict]) -> List[Tuple[str,
str]]:
        """
        檢測實例之間的碰撞

        Args:
            instances: 實例列表

        Returns:
            collisions: 碰撞對列表 [(instance_id1, instance_id2), ...]
        """
        collisions = []

        # 計算每個實例的世界空間 AABB
        world_aabbs = []
        for instance in instances:
            world_aabb = self._compute_world_aabb(instance)
            world_aabbs.append((instance['instance_id'], world_aabb))

        # 兩兩檢測碰撞
        for i in range(len(world_aabbs)):
            for j in range(i + 1, len(world_aabbs)):
                id1, aabb1 = world_aabbs[i]
                id2, aabb2 = world_aabbs[j]

                if self._aabb_intersects(aabb1, aabb2):
                    collisions.append((id1, id2))

        return collisions

    def _compute_world_aabb(self, instance: dict) -> dict:
        """計算實例的世界空間 AABB"""
        # 獲取局部 AABB
        if 'collision_bounds' in instance:
            local_aabb = instance['collision_bounds']
        else:
            # 使用預設 AABB
            local_aabb = {
                'type': 'aabb',
                'min': [-500, -500, 0],
                'max': [500, 500, 2000]
            }

        # 獲取 Transform
        transform = instance['transform']
        translation = transform['translation']

        # 簡化：僅考慮平移（完整實作需要考慮旋轉和縮放）
        world_min = [
            local_aabb['min'][0] + translation['x'],
            local_aabb['min'][1] + translation['y'],
            local_aabb['min'][2] + translation['z']
        ]
        world_max = [
            local_aabb['max'][0] + translation['x'],
            local_aabb['max'][1] + translation['y'],
            local_aabb['max'][2] + translation['z']
        ]

        return {'min': world_min, 'max': world_max}

    def _aabb_intersects(self, aabb1: dict, aabb2: dict) -> bool:
        """檢測兩個 AABB 是否相交"""
        # AABB 相交條件：
        # aabb1.max.x >= aabb2.min.x && aabb1.min.x <= aabb2.max.x &&
        # aabb1.max.y >= aabb2.min.y && aabb1.min.y <= aabb2.max.y &&
        # aabb1.max.z >= aabb2.min.z && aabb1.min.z <= aabb2.max.z

        return (aabb1['max'][0] >= aabb2['min'][0] and aabb1['min'][0] <=
aabb2['max'][0] and
                aabb1['max'][1] >= aabb2['min'][1] and aabb1['min'][1] <=
aabb2['max'][1] and
                aabb1['max'][2] >= aabb2['min'][2] and aabb1['min'][2] <=
aabb2['max'][2])
```

4.3 安全距離檢查

```python
# clearance_checker.py
class ClearanceChecker:
    """安全距離檢查器"""

    def check_clearances(self, instances: List[dict],
                         global_clearance_mm: float = 300.0) -> List[dict]:
        """
        檢查實例之間的安全距離

        Args:
            instances: 實例列表
            global_clearance_mm: 全域安全距離（毫米）

        Returns:
            violations: 違反安全距離的列表
        """
        violations = []

        # 計算每個實例的擴展 AABB（加上安全距離）
        expanded_aabbs = []
        for instance in instances:
            clearance = instance.get('constraints', {}).get('clearance_mm',
global_clearance_mm)
            expanded_aabb = self._expand_aabb(instance, clearance)
            expanded_aabbs.append((instance['instance_id'], expanded_aabb,
clearance))

        # 檢測擴展 AABB 的相交
        detector = AABBCollisionDetector()
        for i in range(len(expanded_aabbs)):
            for j in range(i + 1, len(expanded_aabbs)):
                id1, aabb1, clearance1 = expanded_aabbs[i]
                id2, aabb2, clearance2 = expanded_aabbs[j]

                if detector._aabb_intersects(aabb1, aabb2):
                    # 計算實際距離
                    actual_distance = self._compute_distance(aabb1, aabb2)
                    required_clearance = max(clearance1, clearance2)

                    violations.append({
                        'instance_id1': id1,
                        'instance_id2': id2,
                        'actual_distance_mm': actual_distance,
                        'required_clearance_mm': required_clearance,
                        'violation_mm': required_clearance - actual_distance
                    })

        return violations

    def _expand_aabb(self, instance: dict, clearance_mm: float) -> dict:
        """擴展 AABB（加上安全距離）"""
        world_aabb = AABBCollisionDetector()._compute_world_aabb(instance)

        expanded_min = [
            world_aabb['min'][0] - clearance_mm,
            world_aabb['min'][1] - clearance_mm,
            world_aabb['min'][2] - clearance_mm
        ]
        expanded_max = [
            world_aabb['max'][0] + clearance_mm,
            world_aabb['max'][1] + clearance_mm,
            world_aabb['max'][2] + clearance_mm
        ]

        return {'min': expanded_min, 'max': expanded_max}

    def _compute_distance(self, aabb1: dict, aabb2: dict) -> float:
        """計算兩個 AABB 之間的最小距離"""
        # 簡化：計算中心點之間的距離
        center1 = [
            (aabb1['min'][0] + aabb1['max'][0]) / 2,
            (aabb1['min'][1] + aabb1['max'][1]) / 2,
            (aabb1['min'][2] + aabb1['max'][2]) / 2
        ]
        center2 = [
            (aabb2['min'][0] + aabb2['max'][0]) / 2,
            (aabb2['min'][1] + aabb2['max'][1]) / 2,
            (aabb2['min'][2] + aabb2['max'][2]) / 2
        ]

        distance = np.sqrt(
            (center2[0] - center1[0]) ** 2 +
            (center2[1] - center1[1]) ** 2 +
            (center2[2] - center1[2]) ** 2
        )

        return distance
```

5. NDH 解析器設計

5.1 NDH FDL Parser

```
 # ndh_fdl_parser.py
from typing import List, Dict
import yaml

class NDHFDLParser:
    """NDH FDL 解析器"""

    def __init__(self, asset_library: 'AssetLibrary'):
        self.asset_library = asset_library

    def parse(self, fdl_path: str) -> Dict:
        """
        解析 FDL 檔案並生成 Asset Instance 和 Tag Instance

        Args:
            fdl_path: FDL 檔案路徑

        Returns:
            result: 解析結果
                {
                    'asset_instances': [...],
                    'tag_instances': [...],
                    'connections': [...],
                    'statistics': {...}
                }
        """
        # 讀取 FDL 檔案
        with open(fdl_path, 'r') as f:
            fdl_data = yaml.safe_load(f)

        # 驗證 FDL
        validator = FDLValidator('fdl_v0.1_schema.yaml')
        is_valid, errors = validator.validate(fdl_data)

        if not is_valid:
            raise ValueError(f"Invalid FDL: {errors}")

        # 解析 Asset Instances
        asset_instances = []
        tag_instances = []

        for area in fdl_data['site']['areas']:
            for instance_data in area['instances']:
                # 創建 Asset Instance
                asset_instance = self._create_asset_instance(instance_data, 
area)
                asset_instances.append(asset_instance)

                # 創建 Tag Instances
                tags = self._create_tag_instances(instance_data, 
asset_instance)
                tag_instances.extend(tags)

        # 解析 Connections
        connections = self._parse_connections(fdl_data)

        # 統計資訊
        statistics = {
            'total_asset_instances': len(asset_instances),
            'total_tag_instances': len(tag_instances),
            'total_connections': len(connections),
            'areas': len(fdl_data['site']['areas'])
        }

        return {
            'asset_instances': asset_instances,
            'tag_instances': tag_instances,
            'connections': connections,
            'statistics': statistics
        }

    def _create_asset_instance(self, instance_data: dict, area: dict) -> Dict:
        """創建 Asset Instance"""
        instance_id = instance_data['instance_id']
        ref_asset_id = instance_data['ref_asset']

        # 從 Asset Library 獲取 IADL 資產定義
        asset_definition = self.asset_library.get_asset(ref_asset_id)

        if not asset_definition:
            raise ValueError(f"Asset '{ref_asset_id}' not found in Asset
Library")

        # 創建 Asset Instance
        asset_instance = {
            'instance_id': instance_id,
            'asset_id': ref_asset_id,
            'asset_type': asset_definition['type'],
            'name': instance_data.get('name', instance_id),
            'area': area['name'],
            'transform': instance_data['transform'],
            'metadata': instance_data.get('metadata', {}),
            'constraints': instance_data.get('constraints', {}),
            'collision_bounds': instance_data.get('collision_bounds',

asset_definition.get('collision_bounds', {}))
        }

        return asset_instance

    def _create_tag_instances(self, instance_data: dict,
                             asset_instance: Dict) -> List[Dict]:
        """創建 Tag Instances"""
        tag_instances = []

        # 從 IADL 資產定義獲取 Tags
        asset_definition =
self.asset_library.get_asset(asset_instance['asset_id'])

        if 'tags' not in asset_definition:
            return tag_instances

        # 處理 Tag Overrides
        tag_overrides = {}
        if 'tags_overrides' in instance_data:
            for override in instance_data['tags_overrides']:
                tag_overrides[override['tag_id']] = override

        # 創建 Tag Instances
        for tag_def in asset_definition['tags']:
            tag_id = tag_def['tag_id']

            # 檢查是否有覆寫
            if tag_id in tag_overrides:
                override = tag_overrides[tag_id]
                instance_tag_id = override.get('instance_tag_id',
                                              f"
{asset_instance['instance_id']}_{tag_id}")
                local_transform = override.get('local_transform',
tag_def.get('local_transform', {}))
                properties = {**tag_def.get('properties', {}),
                            **override.get('properties', {})}
                mappings = override.get('mappings', {})
            else:
                instance_tag_id = f"{asset_instance['instance_id']}_{tag_id}"
                local_transform = tag_def.get('local_transform', {})
                properties = tag_def.get('properties', {})
                mappings = {}

            # 創建 Tag Instance
            tag_instance = {
                'instance_tag_id': instance_tag_id,
                'tag_id': tag_id,
                'asset_instance_id': asset_instance['instance_id'],
                'tag_type': tag_def['type'],
                'data_type': tag_def['data_type'],
                'local_transform': local_transform,
                'properties': properties,
                'mappings': mappings
            }

            tag_instances.append(tag_instance)

        return tag_instances

    def _parse_connections(self, fdl_data: dict) -> List[Dict]:
        """解析 Connections"""
        connections = []

        for area in fdl_data['site']['areas']:
            if 'connections' not in area:
                continue

            for conn_data in area['connections']:
                connection = {
                    'connection_id': conn_data['connection_id'],
                    'type': conn_data['type'],
                    'name': conn_data.get('name', ''),
                    'from_instance_id': conn_data['from']['instance_id'],
                    'from_port': conn_data['from'].get('port', ''),
                    'to_instance_id': conn_data['to']['instance_id'],
                    'to_port': conn_data['to'].get('port', ''),
                    'path': conn_data.get('path', {}),
                    'properties': conn_data.get('properties', {})
                }

                connections.append(connection)

        return connections
```

5.2 NDH Asset Servant 生成

```
# ndh_asset_servant_generator.py
class NDHAssetServantGenerator:
    """NDH Asset Servant 生成器"""

    def generate_servants(self, asset_instances: List[Dict],
                         tag_instances: List[Dict]) -> List['AssetServant']:
        """
        生成 Asset Servants

        Args:
            asset_instances: Asset Instance 列表
            tag_instances: Tag Instance 列表

        Returns:
            servants: Asset Servant 列表
        """
        servants = []

        # 按 asset_instance_id 分組 Tag Instances
        tags_by_asset = {}
        for tag in tag_instances:
            asset_id = tag['asset_instance_id']
            if asset_id not in tags_by_asset:
                tags_by_asset[asset_id] = []
            tags_by_asset[asset_id].append(tag)

        # 為每個 Asset Instance 創建 Asset Servant
        for asset_instance in asset_instances:
            instance_id = asset_instance['instance_id']
            tags = tags_by_asset.get(instance_id, [])

            servant = AssetServant(
                instance_id=instance_id,
                asset_id=asset_instance['asset_id'],
                asset_type=asset_instance['asset_type'],
                transform=asset_instance['transform'],
                tags=tags
            )

            servants.append(servant)

        return servants

class AssetServant:
    """Asset Servant (資產服務者)"""

    def __init__(self, instance_id: str, asset_id: str, asset_type: str,
                 transform: dict, tags: List[Dict]):
        self.instance_id = instance_id
        self.asset_id = asset_id
        self.asset_type = asset_type
        self.transform = transform
        self.tags = tags

        # 初始化 Tag 值
        self.tag_values = {}
        for tag in tags:
            self.tag_values[tag['instance_tag_id']] = None

    def update_tag_value(self, instance_tag_id: str, value: any,
                        timestamp: float, quality: str = 'good'):
        """更新 Tag 值"""
        if instance_tag_id not in self.tag_values:
            print(f"[AssetServant] Warning: Tag '{instance_tag_id}' not found")
            return

        self.tag_values[instance_tag_id] = {
            'value': value,
            'timestamp': timestamp,
            'quality': quality
        }

        # 發布事件
        self._publish_tag_value_changed_event(instance_tag_id, value,
timestamp, quality)

    def get_tag_value(self, instance_tag_id: str) -> dict:
        """獲取 Tag 值"""
        return self.tag_values.get(instance_tag_id)

    def _publish_tag_value_changed_event(self, tag_id: str, value: any,
                                        timestamp: float, quality: str):
        """發布 TagValueChanged 事件"""
        event = {
            'event_type': 'TagValueChanged',
            'asset_instance_id': self.instance_id,
            'tag_id': tag_id,
            'value': value,
            'timestamp': timestamp,
            'quality': quality
        }

        # 發布到 Event Bus
        # event_bus.publish(event)
        pass
```

6. 使用範例

6.1 完整的 FDL 範例

```yaml
# demo_plant_a.fdl.yaml
fdl_version: "0.1"

# 全域座標系統與單位制
units:
  length: "mm"
  angle: "deg"
  up_axis: "Z"
  handedness: "right"

# 廠區
site:
  name: "DemoPlantA"
  site_id: "site_001"
  location:
    latitude: 24.7736
    longitude: 120.9436
    altitude: 50.0

  # 區域列表
  areas:
    # 泵房
    - name: "PumpRoom01"
      area_id: "area_pump_room_01"
      type: "production"

      # 區域邊界
      boundary:
        type: "box"
        dimensions:
          width: 10000  # 10m
          depth: 8000   # 8m
          height: 5000  # 5m
        origin:
          x: 0
          y: 0
          z: 0

      # 資產實例
      instances:
        # 泵 A
        - instance_id: "PR01_Pump_A"
          ref_asset: "Pump_Grundfos_CR64_2"
          name: "Pump A"

          transform:
            translation:
              x: 1000
              y: 1000
              z: 0
            rotation:
              x: 0
              y: 0
              z: 0
            scale:
              x: 1.0
              y: 1.0
              z: 1.0

          collision_bounds:
            type: "aabb"
            min: [-400, -400, 0]
            max: [400, 400, 1800]

          constraints:
            clearance_mm: 300
            access_required: true
            fixed: false

          metadata:
            manufacturer: "Grundfos"
            model: "CR 64-2"
            serial_number: "SN123456"
            installation_date: "2024-01-15"

        # 泵 B
        - instance_id: "PR01_Pump_B"
          ref_asset: "Pump_Grundfos_CR64_2"
          name: "Pump B"

          transform:
            translation:
              x: 3000
              y: 1000
              z: 0
            rotation:
              x: 0
              y: 0
              z: 0
            scale:
              x: 1.0
              y: 1.0
              z: 1.0

          collision_bounds:
            type: "aabb"
            min: [-400, -400, 0]
            max: [400, 400, 1800]

          constraints:
            clearance_mm: 300
            access_required: true
            fixed: false

        # 儲水槽
        - instance_id: "PR01_Tank_A"
          ref_asset: "Tank_Vertical_5000L"
          name: "Water Tank A"

          transform:
            translation:
              x: 6000
              y: 3000
              z: 0
            rotation:
              x: 0
              y: 0
              z: 0
            scale:
              x: 1.0
              y: 1.0
              z: 1.0
          collision_bounds:
            type: "aabb"
            min: [-1000, -1000, 0]
            max: [1000, 1000, 3000]

          constraints:
            clearance_mm: 500
            access_required: true
            fixed: true

      # 連接
      connections:
        # 泵 A 到儲水槽的管線
        - connection_id: "conn_pump_a_to_tank_a"
          type: "pipe"
          name: "Pump A Discharge Pipe"

          from:
            instance_id: "PR01_Pump_A"
            port: "discharge"
          to:
            instance_id: "PR01_Tank_A"
            port: "inlet"

          path:
            type: "polyline"
            points:
              - [1000, 1000, 1500]  # 起點（泵 A 出口）
              - [1000, 1000, 2000]  # 向上
              - [6000, 1000, 2000]  # 水平
              - [6000, 3000, 2000]  # 水平
              - [6000, 3000, 1500]  # 向下（儲水槽入口）

          properties:
            diameter_mm: 100
            material: "stainless_steel"
            pressure_rating: "PN16"

          constraints:
            min_bend_radius_mm: 300
            clearance_mm: 100

    # 控制室
    - name: "ControlRoom01"
      area_id: "area_control_room_01"
      type: "control"

      boundary:
        type: "box"
        dimensions:
          width: 6000
          depth: 5000
          height: 3000
        origin:
          x: 15000
          y: 0
          z: 0

      instances:
        # PLC
        - instance id: "CR01 PLC Main"
          ref_asset: "PLC_Siemens_S7_1500"
          name: "Main PLC"

          transform:
            translation:
```

```yaml
            x: 16000
            y: 2500
            z: 1500
          rotation:
            x: 0
            y: 0
            z: 0
          scale:
            x: 1.0
            y: 1.0
            z: 1.0

        tags_overrides:
          - tag_id: "cpu_load"
            instance_tag_id: "CR01_PLC_Main_CPU_Load"
            properties:
              alarm_high: 90.0
              alarm_low: 0.0
            mappings:
              scada_tag: "PLC1.CPU.Load"
              historian_tag: "Plant.ControlRoom.PLC.CPULoad"

# 公用系統
utilities:
  # 電力系統
  - utility_id: "util_electrical_main"
    type: "electrical"
    name: "Main Power Distribution"

    properties:
      voltage: "380V"
      frequency: "50Hz"
      capacity: "500kW"

    connected_instances:
      - instance_id: "PR01_Pump_A"
        connection_point: "power_inlet"
      - instance_id: "PR01_Pump_B"
        connection_point: "power_inlet"
      - instance_id: "CR01_PLC_Main"
        connection_point: "power_inlet"

# 全域約束
global_constraints:
  default_clearance_mm: 300
  min_aisle_width_mm: 1200
  max_stack_height_mm: 5000

  collision_detection:
    enabled: true
    check_aabb: true
    check_clearance: true
    ignore_vertical_clearance: false

  layout_rules:
    - rule_id: "rule_pump_spacing"
      type: "min_distance"
      description: "Pumps must be at least 500mm apart"
      applies_to:
        asset_types: ["Pump"]
      parameters:
        min_distance_mm: 500
```

6.2 使用 FDL

```python
# 使用範例
from ndh_fdl_parser import NDHFDLParser
from fdl_to_usd_composer import FDLToUSDComposer
from collision_detector import AABBCollisionDetector
from clearance_checker import ClearanceChecker

# 1. 解析 FDL
parser = NDHFDLParser(asset_library)
result = parser.parse('demo_plant_a.fdl.yaml')

print(f"Asset Instances: {result['statistics']['total_asset_instances']}")
print(f"Tag Instances: {result['statistics']['total_tag_instances']}")

# 2. 碰撞檢測
detector = AABBCollisionDetector()
collisions = detector.detect_collisions(result['asset_instances'])

if collisions:
    print(f"Found {len(collisions)} collisions:")
    for id1, id2 in collisions:
        print(f"  - {id1} <-> {id2}")

# 3. 安全距離檢查
checker = ClearanceChecker()
violations = checker.check_clearances(result['asset_instances'])

if violations:
    print(f"Found {len(violations)} clearance violations:")
    for v in violations:
        print(f"  - {v['instance_id1']} <-> {v['instance_id2']}: "
              f"violation = {v['violation_mm']} mm")

# 4. 生成 USD 場景
composer = FDLToUSDComposer(
    fdl_data=yaml.safe_load(open('demo_plant_a.fdl.yaml')),
    asset_library_path='/path/to/asset/library'
)
composer.compose('demo_plant_a.usd')

print("USD scene generated: demo_plant_a.usd")

# 5. 生成 Asset Servants
from ndh_asset_servant_generator import NDHAssetServantGenerator

generator = NDHAssetServantGenerator()
servants = generator.generate_servants(
    result['asset_instances'],
    result['tag_instances']
)

print(f"Generated {len(servants)} Asset Servants")
```

7. 實作指南

7.1 實作優先順序

優先級	任務	預估時間
P0	定義 FDL v0.1 JSON Schema	2 天
P0	實作 FDL Validator	2 天
P0	實作 FDL → USD Composer	3 天
P0	實作 AABB 碰撞檢測	2 天
P1	實作 NDH FDL Parser	3 天
P1	實作安全距離檢查	1 天
P1	實作 Asset Servant Generator	2 天
P2	實作 USD Variant 支援	2 天
P2	撰寫單元測試	2 天
P2	撰寫使用者文件	1 天

總計：約 20 天（4 週）

7.2 驗收標準

✅ **功能驗收**： - FDL 檔案可以正確解析 - FDL → USD 轉換生成正確的場景 - 碰撞檢測和安全距離檢查正常工作 - NDH 可以從 FDL 生成 Asset Instance 和 Tag Instance

✅ **品質驗收**： - 所有單元測試通過 - FDL Validator 可以捕獲所有語法和語義錯誤 - USD 場景可以在 Omniverse 中正確顯示 - 程式碼覆蓋率 > 80%

✅ **文件驗收**： - FDL v0.1 規範文件完整 - API 文件完整 - 使用者指南完整

7.3 風險與緩解

風險	影響	機率	緩解措施
FDL Schema 不完整	高	中	與團隊充分討論，參考現有 FDL 範例
USD Reference 效能問題	中	低	使用 USD Instancing 優化
碰撞檢測精度不足	中	中	使用更精確的碰撞檢測演算法（OBB, Mesh）
NDH 解析效能問題	中	低	使用快取和增量更新

總結

本文件定義了 **FDL v0.1** 的完整規範，包括：

1. **FDL Schema 定義** - 明確的語法和語義
2. **FDL → USD 組合策略** - USD Reference + Variant
3. **驗證規則與約束** - Schema 驗證、碰撞檢測、安全距離檢查
4. **NDH 解析器設計** - 從 FDL 生成 Asset Instance 和 Tag Instance
5. **完整的使用範例** - 實際的 FDL 檔案和程式碼範例

核心優勢：

✅ **明確的規範**：語法和語義清晰定義
✅ **可驗證性**：完整的驗證規則和約束
✅ **可組合性**：USD Reference + Variant 策略
✅ **可擴展性**：支援未來功能擴展
✅ **實用性**：完整的實作指南和範例

下一步：

1. 實作 FDL v0.1 JSON Schema
2. 實作 FDL Validator
3. 實作 FDL → USD Composer
4. 整合到 FDL Designer 和 NDH

附錄：

- JSON Schema 文件
- USD Reference 文件
- USD Variant 文件