FDL v0.1 規格更新文件

Factory Description Language (FDL) v0.1 Specification Update

版本: 0.1.1
日期: 2025-10-22
作者: Michael Lin 林志錚
組織: HTFA/Digital Twins Alliance

---

1. 更新概述

本文件針對 FDL v0.1 規格進行增強，新增以下功能：

1. **縮放約束**: 禁止或限制縮放範圍，確保製造業資產的幾何一致性。
2. **碰撞檢測**: 使用 AABB/OBB 快速排除，必要時進行精細網格檢測。
3. **批量佈局**: 支援規則化佈局（網格、直線、圓形），並能輸出 CSV 反查。

---

2. 縮放約束

2.1 問題背景

製造業資產通常具有固定的物理尺寸，不應隨意縮放。非均勻縮放會導致：

- **碰撞檢測錯誤**: AABB/OBB 計算不準確。
- **物理模擬錯誤**: 質量、慣性矩陣等物理屬性失效。
- **視覺不一致**: 資產外觀與實際尺寸不符。

2.2 縮放約束規則

2.2.1 全域縮放約束

在 FDL 的 `global_constraints` 中定義全域縮放約束：

```yaml
global_constraints:
  scaling_constraints:
    allow_scaling: false  # 是否允許縮放（預設 false）
    allow_non_uniform_scaling: false  # 是否允許非均勻縮放（預設 false）
    min_scale: 0.5  # 最小縮放比例（僅在 allow_scaling=true 時有效）
    max_scale: 2.0  # 最大縮放比例（僅在 allow_scaling=true 時有效）
```

2.2.2 資產實例縮放約束

在 Asset Instance 的 `constraints` 中定義實例級別的縮放約束：

```yaml
instances:
  - instance_id: "PR01_Pump_A"
    ref_asset: "Pump_001"
    transform:
      translation: [0, 0, 0]
      rotation: [0, 0, 0]
      scale: [1.0, 1.0, 1.0]  # 必須為 [1.0, 1.0, 1.0]（如果
allow_scaling=false）

    constraints:
      scaling_constraints:
        allow_scaling: false  # 覆蓋全域設定
        allow_non_uniform_scaling: false
        min_scale: 0.8
        max_scale: 1.2
```

2.3 驗證規則

2.3.1 禁止縮放

如果 `allow_scaling = false`，則 `transform.scale` 必須為 `[1.0, 1.0, 1.0]`。

驗證邏輯:

```python
def validate_no_scaling(instance):
    if not instance.constraints.scaling_constraints.allow_scaling:
        scale = instance.transform.scale
        if scale != [1.0, 1.0, 1.0]:
            raise ValidationError(f"Scaling is not allowed for instance
{instance.instance_id}")
```

2.3.2 限制縮放範圍

如果 `allow_scaling = true`，則 `transform.scale` 的每個分量必須在 `[min_scale, max_scale]` 範圍內。

驗證邏輯:

```
def validate_scale_range(instance):
    constraints = instance.constraints.scaling_constraints
    if constraints.allow_scaling:
        scale = instance.transform.scale
        min_s, max_s = constraints.min_scale, constraints.max_scale
        for s in scale:
            if not (min_s <= s <= max_s):
                raise ValidationError(f"Scale {s} out of range [{min_s},
{max_s}]")
```

2.3.3 禁止非均勻縮放

如果 `allow_non_uniform_scaling = false`，則 `transform.scale` 的三個分量必須相等。

驗證邏輯:

```
def validate_uniform_scaling(instance):
    constraints = instance.constraints.scaling_constraints
    if not constraints.allow_non_uniform_scaling:
        scale = instance.transform.scale
        if not (scale[0] == scale[1] == scale[2]):
            raise ValidationError(f"Non-uniform scaling is not allowed for
instance {instance.instance_id}")
```

---

3. 碰撞檢測

3.1 碰撞檢測策略

碰撞檢測分為兩個階段：

1. **Broadphase（粗檢）**：使用 AABB/OBB 快速排除不可能碰撞的物體對。
2. **Narrowphase（細檢）**：對可能碰撞的物體對進行精細網格檢測。

3.2 AABB/OBB 碰撞檢測

3.2.1 AABB（軸對齊包圍盒）

定義:

AABB 是與座標軸對齊的矩形包圍盒，由最小點 `min` 和最大點 `max` 定義。

碰撞檢測:

兩個 AABB 碰撞當且僅當它們在所有三個軸上都有重疊。

```
def aabb_collision(aabb1, aabb2):
    """檢測兩個 AABB 是否碰撞"""
    return (aabb1.min.x <= aabb2.max.x and aabb1.max.x >= aabb2.min.x and
            aabb1.min.y <= aabb2.max.y and aabb1.max.y >= aabb2.min.y and
            aabb1.min.z <= aabb2.max.z and aabb1.max.z >= aabb2.min.z)
```

優點: - 計算簡單，速度快。 - 適合作為 Broadphase 快速排除。

缺點: - 對旋轉的物體不精確，可能產生誤報。

3.2.2 OBB（定向包圍盒）

定義:

OBB 是可以旋轉的矩形包圍盒，由中心點、三個軸向量和三個半徑定義。

碰撞檢測:

使用分離軸定理（Separating Axis Theorem, SAT）檢測兩個 OBB 是否碰撞。

```
def obb_collision(obb1, obb2):
    """檢測兩個 OBB 是否碰撞（使用 SAT）"""
    # 實作分離軸定理
    # 詳細實作見參考文獻
    pass
```

優點: - 對旋轉的物體更精確。 - 仍然比網格檢測快得多。

缺點: - 計算比 AABB 複雜。

3.3 網格碰撞檢測

對於需要高精度碰撞檢測的場景，使用網格級別的碰撞檢測。

工具: - **trimesh**: Python 的網格處理庫，支援網格碰撞檢測。 - **PyBullet**: 物理引擎，支援複雜的碰撞檢測和物理模擬。

範例（使用 trimesh）:

```
import trimesh

def mesh_collision(mesh1, transform1, mesh2, transform2):
    """檢測兩個網格是否碰撞"""
    # 應用變換
    mesh1_transformed = mesh1.copy()
    mesh1_transformed.apply_transform(transform1)

    mesh2_transformed = mesh2.copy()
    mesh2_transformed.apply_transform(transform2)

    # 使用 trimesh 的碰撞檢測
    collision_manager = trimesh.collision.CollisionManager()
    collision_manager.add_object('mesh1', mesh1_transformed)
    collision_manager.add_object('mesh2', mesh2_transformed)

    is_collision, _ =
collision_manager.in_collision_internal(return_names=True)
    return is_collision
```

3.4 碰撞檢測配置

在 FDL 的 `global_constraints` 中配置碰撞檢測：

```
global_constraints:
  collision_detection:
    enabled: true
    broadphase: "aabb"  # aabb, obb
    narrowphase: "mesh"  # mesh, none
    check_clearance: true  # 是否檢查安全距離
    default_clearance_mm: 300  # 預設安全距離（毫米）
    ignore_vertical_clearance: false  # 是否忽略垂直方向的安全距離
```

3.5 碰撞檢測流程

```
def detect_collisions(instances, config):
    """檢測所有實例之間的碰撞"""
    collisions = []

    # Broadphase: 使用 AABB/OBB 快速排除
    potential_collisions = []
    for i in range(len(instances)):
        for j in range(i + 1, len(instances)):
            inst1, inst2 = instances[i], instances[j]

            if config.broadphase == "aabb":
                if aabb_collision(inst1.aabb, inst2.aabb):
                    potential_collisions.append((inst1, inst2))
            elif config.broadphase == "obb":
                if obb_collision(inst1.obb, inst2.obb):
                    potential_collisions.append((inst1, inst2))

    # Narrowphase: 對可能碰撞的物體對進行精細檢測
    if config.narrowphase == "mesh":
        for inst1, inst2 in potential_collisions:
            if mesh_collision(inst1.mesh, inst1.transform, inst2.mesh,
inst2.transform):
                collisions.append((inst1, inst2))
    else:
        # 如果不進行 Narrowphase，則 Broadphase 的結果即為最終結果
        collisions = potential_collisions

    # 檢查安全距離
    if config.check_clearance:
        for inst1, inst2 in collisions:
            clearance = calculate_clearance(inst1, inst2)
            if clearance < config.default_clearance_mm:
                print(f"Warning: Clearance {clearance}mm <
{config.default_clearance_mm}mm")

    return collisions
```

---

4. 批量佈局

4.1 批量佈局需求

在工業場景中，經常需要批量佈置相同或相似的資產，例如：

- **網格佈局**: 在矩形區域內均勻分佈資產（如儲罐陣列）。
- **直線佈局**: 沿直線排列資產（如管道閥門）。
- **圓形佈局**: 沿圓周排列資產（如圓形配電盤）。

4.2 批量佈局配置

在 FDL 中定義批量佈局規則：

```yaml
batch_layouts:
  - layout_id: "layout_001"
    name: "Tank Array"
    pattern: "grid"  # grid, line, circle
    ref_asset: "Tank_001"  # 引用的 IADL 資產 ID

    # 網格佈局參數
    grid_params:
      rows: 3
      columns: 4
      spacing_x: 2000  # mm
      spacing_y: 2000  # mm
      origin: [0, 0, 0]  # 起始位置（相對於 Area 原點）

    # 實例命名規則
    naming:
      prefix: "Tank"
      suffix_format: "{row:02d}_{col:02d}"  # 例如：Tank_01_01, Tank_01_02

    # 輸出 CSV
    export_csv: true
    csv_path: "/output/tank_array.csv"
```

4.3 批量佈局模式

4.3.1 網格佈局（Grid）

參數: - `rows`：行數 - `columns`：列數 - `spacing_x`：X 方向間距（mm） - `spacing_y`：Y 方向間距（mm） - `origin`：起始位置

生成邏輯:

```python
def generate_grid_layout(params):
    """生成網格佈局的實例列表"""
    instances = []
    for row in range(params.rows):
        for col in range(params.columns):
            instance_id = f"{params.naming.prefix}_{row:02d}_{col:02d}"
            translation = [
                params.origin[0] + col * params.spacing_x,
                params.origin[1] + row * params.spacing_y,
                params.origin[2]
            ]
            instances.append({
                "instance_id": instance_id,
                "ref_asset": params.ref_asset,
                "transform": {
                    "translation": translation,
                    "rotation": [0, 0, 0],
                    "scale": [1.0, 1.0, 1.0]
                }
            })
    return instances
```

4.3.2 直線佈局（Line）

參數: - `count`：實例數量 - `start`：起始位置 - `end`：結束位置

生成邏輯:

```python
def generate_line_layout(params):
    """生成直線佈局的實例列表"""
    instances = []
    start = np.array(params.start)
    end = np.array(params.end)
    direction = (end - start) / (params.count - 1)

    for i in range(params.count):
        instance_id = f"{params.naming.prefix}_{i:02d}"
        translation = start + i * direction
        instances.append({
            "instance_id": instance_id,
            "ref_asset": params.ref_asset,
            "transform": {
                "translation": translation.tolist(),
                "rotation": [0, 0, 0],
                "scale": [1.0, 1.0, 1.0]
            }
        })
    return instances
```

4.3.3 圓形佈局（Circle）

參數: - `count`：實例數量 - `center`：圓心位置 - `radius`：半徑（mm） - `start_angle`：起始角度（度）

生成邏輯:

```python
def generate_circle_layout(params):
    """生成圓形佈局的實例列表"""
    instances = []
    center = np.array(params.center)
    angle_step = 360.0 / params.count

    for i in range(params.count):
        instance_id = f"{params.naming.prefix}_{i:02d}"
        angle = params.start_angle + i * angle_step
        angle_rad = np.radians(angle)

        translation = center + np.array([
            params.radius * np.cos(angle_rad),
            params.radius * np.sin(angle_rad),
            0
        ])

        instances.append({
            "instance_id": instance_id,
            "ref_asset": params.ref_asset,
            "transform": {
                "translation": translation.tolist(),
                "rotation": [0, 0, angle],  # 面向圓心
                "scale": [1.0, 1.0, 1.0]
            }
        })
    return instances
```

4.4 CSV 輸出

批量佈局生成的實例可以輸出為 CSV 文件，便於反查和管理。

CSV 格式:

```
 instance_id,ref_asset,translation_x,translation_y,translation_z,rotation_x,rotati
Tank 00 00,Tank 001,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0
Tank_00_01,Tank_001,2000.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0
Tank_00_02,Tank_001,4000.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0
...
```

生成邏輯:

```python
import csv

def export_to_csv(instances, csv_path):
    """將實例列表輸出為 CSV 文件"""
    with open(csv_path, 'w', newline='') as csvfile:
        fieldnames = ['instance_id', 'ref_asset',
                      'translation_x', 'translation_y', 'translation_z',
                      'rotation_x', 'rotation_y', 'rotation_z',
                      'scale_x', 'scale_y', 'scale_z']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()
        for inst in instances:
            writer.writerow({
                'instance_id': inst['instance_id'],
                'ref_asset': inst['ref_asset'],
                'translation_x': inst['transform']['translation'][0],
                'translation_y': inst['transform']['translation'][1],
                'translation_z': inst['transform']['translation'][2],
                'rotation_x': inst['transform']['rotation'][0],
                'rotation_y': inst['transform']['rotation'][1],
                'rotation_z': inst['transform']['rotation'][2],
                'scale_x': inst['transform']['scale'][0],
                'scale_y': inst['transform']['scale'][1],
                'scale_z': inst['transform']['scale'][2],
            })
```

---

5. 完整範例

5.1 FDL 文件範例（包含所有新功能）

```yaml
fdl_version: "0.1.1"

units:
  length: "mm"
  angle: "deg"
  up_axis: "Z"
  handedness: "right"

global_constraints:
  # 縮放約束
  scaling_constraints:
    allow_scaling: false
    allow_non_uniform_scaling: false
    min_scale: 0.5
    max_scale: 2.0

  # 碰撞檢測
  collision_detection:
    enabled: true
    broadphase: "aabb"
    narrowphase: "mesh"
    check_clearance: true
    default_clearance_mm: 300
    ignore_vertical_clearance: false

site:
  name: "DemoPlantA"
  site_id: "site_001"

  areas:
    - name: "TankFarm"
      area_id: "area_001"
      type: "storage"

      # 手動定義的實例
      instances:
        - instance_id: "TF_Pump_A"
          ref asset: "Pump_001"
          transform:
            translation: [0, 0, 0]
            rotation: [0, 0, 0]
            scale: [1.0, 1.0, 1.0]

          constraints:
            scaling_constraints:
              allow_scaling: false

# 批量佈局
batch layouts:
  - layout_id: "layout_001"
    name: "Tank Array"
    pattern: "grid"
    ref_asset: "Tank_001"

    grid_params:
      rows: 3
      columns: 4
```

```yaml
    spacing_x: 2000
    spacing_y: 2000
    origin: [5000, 0, 0]

  naming:
    prefix: "Tank"
    suffix_format: "{row:02d}_{col:02d}"

  export_csv: true
  csv_path: "/output/tank_array.csv"
```

---

6. 實作指南

6.1 驗證器實作

```
class FDLValidator:
    def __init__(self, fdl_data):
        self.fdl_data = fdl_data
        self.errors = []

    def validate(self):
        """執行所有驗證"""
        self.validate_scaling_constraints()
        self.validate_collision_detection()
        return len(self.errors) == 0

    def validate_scaling_constraints(self):
        """驗證縮放約束"""
        global_constraints = self.fdl_data.get('global_constraints', {})
        scaling_constraints = global_constraints.get('scaling_constraints', {})

        for area in self.fdl_data['site']['areas']:
            for instance in area.get('instances', []):
                self._validate_instance_scaling(instance, scaling_constraints)

    def _validate_instance_scaling(self, instance, global_constraints):
        """驗證單個實例的縮放"""
        # 獲取實例級別的約束（如果有）
        inst_constraints = instance.get('constraints',
{}).get('scaling_constraints', {})
        constraints = {**global_constraints, **inst_constraints}

        scale = instance['transform']['scale']

        # 檢查是否允許縮放
        if not constraints.get('allow_scaling', False):
            if scale != [1.0, 1.0, 1.0]:
                self.errors.append(f"Scaling not allowed for
{instance['instance_id']}")

        # 檢查縮放範圍
        if constraints.get('allow_scaling', False):
            min_s = constraints.get('min_scale', 0.5)
            max_s = constraints.get('max_scale', 2.0)
            for s in scale:
                if not (min_s <= s <= max_s):
                    self.errors.append(f"Scale {s} out of range for
{instance['instance_id']}")

        # 檢查是否允許非均勻縮放
        if not constraints.get('allow_non_uniform_scaling', False):
            if not (scale[0] == scale[1] == scale[2]):
                self.errors.append(f"Non-uniform scaling not allowed for
{instance['instance_id']}")

    def validate_collision_detection(self):
        """驗證碰撞檢測"""
        # 實作碰撞檢測邏輯
        pass
```

6.2 批量佈局生成器實作

```python
class BatchLayoutGenerator:
    def __init__(self, fdl_data):
        self.fdl_data = fdl_data

    def generate_all_layouts(self):
        """生成所有批量佈局"""
        batch_layouts = self.fdl_data.get('batch_layouts', [])
        all_instances = []

        for layout in batch_layouts:
            instances = self.generate_layout(layout)
            all_instances.extend(instances)

            # 輸出 CSV
            if layout.get('export_csv', False):
                export_to_csv(instances, layout['csv_path'])

        return all_instances

    def generate_layout(self, layout):
        """根據模式生成佈局"""
        pattern = layout['pattern']

        if pattern == 'grid':
            return generate_grid_layout(layout)
        elif pattern == 'line':
            return generate_line_layout(layout)
        elif pattern == 'circle':
            return generate_circle_layout(layout)
        else:
            raise ValueError(f"Unknown pattern: {pattern}")
```

---

7. 參考文件

- IADL v1.0 規格文件
- FDL v0.1 規格文件
- 軟體設計文件 v2.1
- trimesh 文件
- PyBullet 文件
- 分離軸定理（SAT）