2R1 Analysis

In [1]:
```python
import ifcopenshell
import ifcopenshell.geom
import math
import pandas as pd

# Load the IFC file
base_path = r"C:\open3d-env\Generative design\Database"
ifc_file = ifcopenshell.open(f"{base_path}/LOD300/LOD300/2R1.ifc")

# Function to calculate only the top face area of the slab
def calculate_top_face_area(slab):
    """Extract geometry and calculate the area of only the top face of the slab."""
    try:
        # Generate geometry settings
        settings = ifcopenshell.geom.settings()
        settings.set(settings.USE_WORLD_COORDS, True)

        # Create the geometry for the slab
        geom = ifcopenshell.geom.create_shape(settings, slab)

        # Extract vertices and faces from geometry
        shape = geom.geometry
        vertices = shape.verts
        faces = shape.faces

        total_area = 0.0

        # Loop through triangular faces
        for i in range(0, len(faces), 3):
            # Get the three vertices of the triangle
            v1 = vertices[faces[i] * 3:faces[i] * 3 + 3]
            v2 = vertices[faces[i+1] * 3:faces[i+1] * 3 + 3]
            v3 = vertices[faces[i+2] * 3:faces[i+2] * 3 + 3]

            # Calculate the two edges of the triangle
            a = [v2[j] - v1[j] for j in range(3)]
            b = [v3[j] - v1[j] for j in range(3)]

            # Compute the cross product (normal vector)
            cross_product = [
                a[1] * b[2] - a[2] * b[1],
                a[2] * b[0] - a[0] * b[2],
                a[0] * b[1] - a[1] * b[0]
            ]

            # Compute the Z-component of the normal
            normal_z = cross_product[2]

            # Include only upward-facing triangles (Z-component > 0)
            if normal_z > 0:
                triangle_area = 0.5 * math.sqrt(sum(c ** 2 for c in cross_product))
                total_area += triangle_area

        return total_area
    except Exception as e:
        print(f"Error calculating geometry for slab {slab.GlobalId}: {e}")
        return None

# Get all slabs
slabs = ifc_file.by_type("IfcSlab")
print(f"Found {len(slabs)} slabs in the model")

# Process slabs
slab_data = []
for slab in slabs:
    # Calculate the area for the top face only
    area = calculate_top_face_area(slab)

    # Store slab information
    slab_data.append({
        'GlobalId': slab.GlobalId,
        'Name': slab.Name if hasattr(slab, 'Name') else None,
        'Type': slab.PredefinedType if hasattr(slab, 'PredefinedType') else None,
        'TopFaceArea': area
    })

# Create DataFrame
df_slabs = pd.DataFrame(slab_data)

# Filter slabs for Cast in-situ Slabs
cast_in_situ_slabs = df_slabs[
    (df_slabs['Type'] == 'FLOOR') &
    (df_slabs['Name'].str.contains('Cast in-situ Slab', case=False, na=False))
```

```
]

# Calculate total area
total_area = cast_in_situ_slabs['TopFaceArea'].sum()

# Print results
print("\nCast in-situ Slab Top Face Areas:")
print("-" * 50)
for idx, row in cast_in_situ_slabs.iterrows():
    print(f"Slab: {row['Name']}, Top Face Area: {row['TopFaceArea']:.2f} m²")

print("\nTotal Cast in-situ Slab Top Face Area:")
print("-" * 50)
print(f"{total_area:.2f} m²")

# Export results to CSV
cast_in_situ_slabs.to_csv('cast_in_situ_slab_top_face_analysis.csv', index=False)
print("\nDetailed Cast in-situ Slab top face analysis exported to 'cast_in_situ_slab_top_face_analysis.csv'")
```

```
Found 18 slabs in the model

Cast in-situ Slab Top Face Areas:
--------------------------------------------------
Slab: Floor:S_Cast in-situ Slab_175:3192670, Top Face Area: 0.37 m²
Slab: Floor:S_Cast in-situ Slab_175:3192524, Top Face Area: 9.54 m²
Slab: Floor:S_Cast in-situ Slab_150:3192510, Top Face Area: 5.58 m²
Slab: Floor:S_Cast in-situ Slab_150:3192516, Top Face Area: 9.96 m²
Slab: Floor:S_Cast in-situ Slab_175:3192530, Top Face Area: 10.49 m²
Slab: Floor:S_Cast in-situ Slab_175:3197663, Top Face Area: 2.23 m²
Slab: Floor:S_Cast in-situ Slab_175:3197672, Top Face Area: 3.23 m²

Total Cast in-situ Slab Top Face Area:
--------------------------------------------------
41.40 m²

Detailed Cast in-situ Slab top face analysis exported to 'cast_in_situ_slab_top_face_analysis.csv'
```

In [2]:

```
import ifcopenshell
import ifcopenshell.geom
import math
import pandas as pd
import numpy as np
from pathlib import Path
import seaborn as sns
import matplotlib.pyplot as plt
from collections import defaultdict

class CarbonCalculator:
    def __init__(self):
        # Default dimensions in meters
        self.default_dimensions = {
            'IfcWall': {'thickness': 0.1, 'height': 2.8},
            'IfcWallStandardCase': {'thickness': 0.1, 'height': 2.8},
            'IfcSlab': {'thickness': 0.175},  # Based on naming convention seen (Slab_175)
            'IfcColumn': {'width': 0.3, 'depth': 0.3, 'height': 2.8},
            'IfcBeam': {'width': 0.25, 'depth': 0.4, 'height': 0.4},
            'IfcWindow': {'thickness': 0.05, 'height': 1.5},
            'IfcDoor': {'thickness': 0.05, 'height': 2.1}
        }

        # Carbon coefficients in kgCO2e/kg
        self.carbon_coefficients = {
            'CONCRETE': 0.15,    # Generic concrete C30
            'STEEL': 2.34,       # Structural steel
            'GLASS': 2.47,       # Double Glazed Glass
            'WOOD': -1.31,       # Cross_Laminated_Timber
            'BRICK': 0.21,       # Clay brick
            'ALUMINUM': 9.05,    # Sheet_Aluminium_Virgin
            'PLASTER': 0.13,     # Gypsum plaster
            'INSULATION': 2.55,  # Generic insulation
            'REBAR': 2.34,       # Steel reinforcement
            'TILE': 0.78,        # Ceramic tile
            'PAINT': 0.87,       # Paint coating
            'LAMINATED_WOOD': -1.31  # Using same as CLT for now
        }

        # Material densities in kg/m³
        self.material_densities = {
            'CONCRETE': 2400,
            'STEEL': 7850,
            'GLASS': 2500,
            'WOOD': 500,
            'BRICK': 1800,
            'ALUMINUM': 2700,
            'PLASTER': 1200,
            'INSULATION': 30,
            'REBAR': 7850,
            'TILE': 2000,        # Ceramic tile density
            'PAINT': 1500,       # Paint coating density
```

```python
            'LAMINATED_WOOD': 480  # Typical laminated wood density
        }

        # Material thicknesses in meters for surface finishes
        self.material_thicknesses = {
            'TILE': 0.005,        # 5mm for tiles
            'PAINT': 0.001,      # 1mm for paint
            'LAMINATED_WOOD': 0.01  # 10mm for laminated wood flooring
        }

    def get_material_properties(self, material):
        """Get carbon coefficient and density for a material"""
        material_upper = material.upper()
        density = self.material_densities.get(material_upper, self.material_densities['CONCRETE'])
        carbon_coeff = self.carbon_coefficients.get(material_upper, self.carbon_coefficients['CONCRETE'])
        return carbon_coeff, density

    def calculate_carbon_footprint(self, volume_m3, material):
        """Calculate carbon footprint for given volume and material"""
        carbon_coeff, density = self.get_material_properties(material)
        mass = volume_m3 * density  # kg
        return mass * carbon_coeff  # kgCO2e

class IFCAnalyzer:
    def __init__(self, ifc_file_path):
        self.ifc_file = ifcopenshell.open(ifc_file_path)
        self.element_types = [
            'IfcBeam', 'IfcColumn', 'IfcDoor', 'IfcSlab',
            'IfcWall', 'IfcWallStandardCase', 'IfcWindow'
        ]
        self.carbon_calculator = CarbonCalculator()
        self.data = self._collect_data()

    def calculate_top_face_area(self, slab):
        """Extract geometry and calculate the area of only the top face of the slab."""
        try:
            # Generate geometry settings
            settings = ifcopenshell.geom.settings()
            settings.set(settings.USE_WORLD_COORDS, True)

            # Create the geometry for the slab
            geom = ifcopenshell.geom.create_shape(settings, slab)

            # Extract vertices and faces from geometry
            shape = geom.geometry
            vertices = shape.verts
            faces = shape.faces

            total_area = 0.0

            # Loop through triangular faces
            for i in range(0, len(faces), 3):
                # Get the three vertices of the triangle
                v1 = vertices[faces[i] * 3:faces[i] * 3 + 3]
                v2 = vertices[faces[i+1] * 3:faces[i+1] * 3 + 3]
                v3 = vertices[faces[i+2] * 3:faces[i+2] * 3 + 3]

                # Calculate the two edges of the triangle
                a = [v2[j] - v1[j] for j in range(3)]
                b = [v3[j] - v1[j] for j in range(3)]

                # Compute the cross product (normal vector)
                cross_product = [
                    a[1] * b[2] - a[2] * b[1],
                    a[2] * b[0] - a[0] * b[2],
                    a[0] * b[1] - a[1] * b[0]
                ]

                # Compute the Z-component of the normal
                normal_z = cross_product[2]

                # Include only upward-facing triangles (Z-component > 0)
                if normal_z > 0:
                    triangle_area = 0.5 * math.sqrt(sum(c ** 2 for c in cross_product))
                    total_area += triangle_area

            return total_area
        except Exception as e:
            print(f"Error calculating geometry for slab {slab.GlobalId}: {e}")
            return None

    def get_material_info(self, element):
        materials = []
        if element.HasAssociations:
            for association in element.HasAssociations:
                if association.is_a('IfcRelAssociatesMaterial'):
                    material = association.RelatingMaterial
                    if material.is_a('IfcMaterial'):
                        materials.append(material.Name)
```

```python
                elif material.is_a('IfcMaterialLayer'):
                    materials.append(material.Material.Name)
                elif material.is_a('IfcMaterialLayerSet'):
                    for layer in material.MaterialLayers:
                        if layer.Material:
                            materials.append(layer.Material.Name)
                elif material.is_a('IfcMaterialList'):
                    for mat in material.Materials:
                        materials.append(mat.Name)

        # Check element type and name for additional materials
        element_type = element.is_a()
        element_name = element.Name if hasattr(element, 'Name') else ""

        if element_type == 'IfcSlab':
            if 'Laminated_Wood' in element_name:
                materials.append('WOOD')
            if 'Tile' in element_name:
                materials.append('TILE')

        elif element_type in ['IfcWall', 'IfcWallStandardCase']:
            if 'Paint' in element_name:
                materials.append('PAINT')
            if 'Tile' in element_name:
                materials.append('TILE')

        # Return list of unique materials
        return list(set(materials)) if materials else ['CONCRETE']

    def calculate_volume(self, element, material):
        """Calculate volume based on element type, geometry and material"""
        element_type = element.is_a()

        if element_type == 'IfcSlab':
            top_area = self.calculate_top_face_area(element)
            if not top_area:
                top_area = 1.0  # Default 1m² if area calculation fails

            # Handle different materials
            if material.upper() in ['TILE', 'PAINT', 'LAMINATED_WOOD']:
                thickness = self.carbon_calculator.material_thicknesses[material.upper()]
                return top_area * thickness
            else:
                # For structural materials (concrete, etc)
                return top_area * self.carbon_calculator.default_dimensions['IfcSlab']['thickness']

        else:
            # Use default dimensions for other element types
            default_dims = self.carbon_calculator.default_dimensions[element_type]
            if element_type in ['IfcWall', 'IfcWallStandardCase']:
                return default_dims['thickness'] * default_dims['height'] * 1.0
            elif element_type in ['IfcBeam', 'IfcColumn']:
                return default_dims['width'] * default_dims['depth'] * default_dims['height']
            else:
                return default_dims.get('thickness', 0.1) * default_dims.get('height', 3.0) * 1.0

    def _collect_data(self):
        data = []
        for element_type in self.element_types:
            elements = self.ifc_file.by_type(element_type)
            for element in elements:
                materials = self.get_material_info(element)
                element_name = element.Name if hasattr(element, 'Name') else None

                # Process each material for the element
                for material in materials:
                    volume = self.calculate_volume(element, material)
                    carbon_footprint = self.carbon_calculator.calculate_carbon_footprint(volume, material)

                    data.append({
                        'type': element_type,
                        'material': material,
                        'volume_m3': volume,
                        'carbon_footprint': carbon_footprint,
                        'id': element.GlobalId,
                        'name': element_name
                    })
        return pd.DataFrame(data)

    def analyze_carbon_footprint(self):
        carbon_analysis = self.data.groupby(['type', 'material']).agg({
            'carbon_footprint': 'sum',
            'volume_m3': 'sum'
        }).round(4)

        return carbon_analysis

    def generate_carbon_report(self):
        print("Carbon Footprint Analysis Report")
```

```python
        print("=" * 50)

        # Get analysis data
        carbon_analysis = self.analyze_carbon_footprint()
        total_carbon = self.data['carbon_footprint'].sum()

        # Create a more readable format
        print("\nMaterial Analysis by Element Type:")
        print("-" * 50)
        print(f"{'Element Type':<20} {'Material':<15} {'Volume (m³)':<12} {'Carbon (kgCO2e)'}")
        print("-" * 80)

        # Sort by element type and material
        for (element_type, material), row in carbon_analysis.iterrows():
            print(f"{element_type:<20} {material:<15} {row['volume_m3']:>11.2f} {row['carbon_footprint']:>14.2f}")

        print("\nSummary:")
        print("-" * 50)
        print(f"Total Project Carbon Footprint: {total_carbon:.2f} kgCO2e")

        # Calculate percentage contribution by element type
        type_totals = self.data.groupby('type')['carbon_footprint'].sum()
        print("\nCarbon Footprint Distribution by Element Type:")
        print("-" * 50)
        for element_type, carbon in type_totals.items():
            percentage = (carbon / total_carbon) * 100
            print(f"{element_type:<20} {carbon:>10.2f} kgCO2e ({percentage:>6.1f}%)")

if __name__ == "__main__":
    base_path = Path(r"C:\open3d-env\Generative design\Database")
    ifc_path = base_path / "LOD300" / "LOD300" / "2R1.ifc"

    try:
        analyzer = IFCAnalyzer(ifc_path)
        analyzer.generate_carbon_report()

    except Exception as e:
        print(f"Error processing IFC file: {e}")
```

```
Carbon Footprint Analysis Report
==================================================

Material Analysis by Element Type:
--------------------------------------------------
Element Type         Material        Volume (m³)  Carbon (kgCO2e)
--------------------------------------------------------------------------------
IfcBeam              Concrete, Cast In Situ        0.36         129.60
IfcColumn            Concrete - Cast-in-Place Concrete        1.51         544.32
IfcDoor              <Unnamed>              0.10          37.80
IfcDoor              Aluminum               0.10        2565.68
IfcDoor              Default Frame          0.10          37.80
IfcDoor              Door - Frame/Mullion        0.10          37.80
IfcDoor              Door - Glazing         0.10          37.80
IfcDoor              Door - Handle          0.10          37.80
IfcDoor              Door - Panel           0.10          37.80
IfcDoor              Door Handle            0.21          75.60
IfcDoor              Door Panel             0.10          37.80
IfcDoor              Metal - Aluminium        0.10          37.80
IfcDoor              Mild Steel             0.10          37.80
IfcDoor              PVC, Flexible          0.10          37.80
IfcDoor              PVC, Flexible(1)        0.10          37.80
IfcSlab              CONCRETE               7.24        2608.13
IfcSlab              TILE                   0.06          92.58
IfcSlab              WOOD                   4.11       -2689.33
IfcWall              CONCRETE              13.16        4737.60
IfcWall              TILE                   4.48        6988.80
IfcWallStandardCase  CONCRETE              12.88        4636.80
IfcWallStandardCase  TILE                   4.20        6552.00
IfcWindow            <Unnamed>              0.15          54.00
IfcWindow            Aluminium              0.15          54.00
IfcWindow            Aluminum, Gray         0.15          54.00
IfcWindow            Glass                  0.30        1852.50

Summary:
--------------------------------------------------
Total Project Carbon Footprint: 28672.07 kgCO2e

Carbon Footprint Distribution by Element Type:
--------------------------------------------------
IfcBeam                  129.60 kgCO2e (   0.5%)
IfcColumn                544.32 kgCO2e (   1.9%)
IfcDoor                 3057.08 kgCO2e (  10.7%)
IfcSlab                   11.38 kgCO2e (   0.0%)
IfcWall                11726.40 kgCO2e (  40.9%)
IfcWallStandardCase    11188.80 kgCO2e (  39.0%)
IfcWindow               2014.50 kgCO2e (   7.0%)
```

In [ ]: