Apartment Finder: Workflow Orchestration Example

Leeon Israel

Problem Statement. Users need to evaluate apartments across multiple criteria. Each criterion requires different analysis. Need an automated, intelligent ranking system.

Solution:
Orchestrated Agent Workflow

At a glance

- 5 specialized agents working together
- 3 orchestration patterns:
    - Sequential
    - Branch
    - Aggregate
- Result: Ranked apartment recommendations with scores

User Input → Filter → [Score A, Score B] → Rank → Output

Project Architecture

```
C3AN_Autonomy/
|___ Code/
|      |___ Agents/              # 5 apartment agents
|      |___ Assets/
|            |___ Resources/     # Config files
|            |___ Tools/         # Shared utilities
|___ Data/
|      |___ Primary/             # Input: preferences
|      |___ Outputs/             # Output: rankings
|___ Workflow/
|      |___ Engine/              # Orchestration operators
|      |___ Apartment_Finder/   # Pipeline
|___ Knowledge/                  # Schemas & knowledge
graph
```

Core Orchestration Operators

```python
def run(name: str, fn: Callable, *args, **kwargs):
    out = fn(*args, **kwargs)
    assert isinstance(out, Mapping)  # Must return dict
    return out
```

01

Sequential (seq) Operator

- Pattern: One input → One output
- Use case: Linear data transformation
- Contract: Must return a dictionary

```python
def run(parent_name: str, branches: list[dict]):
    outs = {}
    for branch in branches:
        name = branch["name"]  # Must be unique
        out = branch["fn"](*args, **kwargs)
        assert isinstance(out, Mapping)  # Must return dict
        outs[name] = out
    return outs  # Returns dict of {name: output}
```

02

Branch (brn) Operator

- Pattern: One input → Multiple parallel outputs
- Use case: Independent parallel computations
- Contract: All branches return mappins; names must be unique

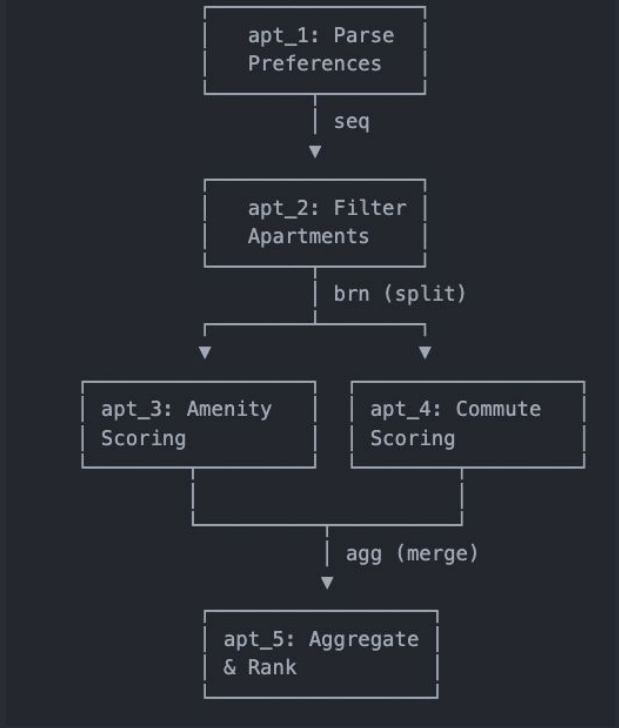
```python
def run(name: str, fn: Callable, inputs: dict, **kwargs):
    # inputs: {key: upstream_output, ...}
    out = fn(**inputs, **kwargs)
    assert isinstance(out, Mapping)  # Must return dict
    return out
```

03

Aggregate (agg) Operator

- Pattern: Multiple inputs → One merged output
- Use case: Combine results from parallel branches
- Contract: Accepts named inputs dict, returns mapping

Apartment Finder Workflow Architecture



Agent 1

Parse User Preferences (seq)

Input: `user_preferences.json`

```
{
  "user_id": "leo_001",
  "max_budget": 1800,
  "location": "downtown",
  "min_bedrooms": 1,
  "required_amenities": ["parking", "gym", "pets"],
  "max_commute_miles": 5.0
}
```

Function:

```
def run(prefs_path: str) -> dict:
    # Load JSON, validate, and structure preferences
    return {
        "user_id": ...,
        "preferences": {...}
    }
```

Output: Structured preferences dictionary

Agent 2

Filter Apartments (seq)

Key Metrics:

- Total apartments: 8
- Filtered apartment: 6 (75% pass rate)

Input:

- User preferences (from Agent 1)
- Apartment listings JSON (8 apartments)

Logic:

```
filtered = []
for apt in all_apartments:
    if (apt["rent"] <= max_budget and
        apt["location"] == desired_location and
        apt["bedrooms"] >= min_bedrooms):
        filtered.append(apt)
```

Output: 6 apartments that meet criteria

Branch Operator: Parallel Scoring(brn)

Why Branch:

- Amenity scoring and commute scoring are independent
- Can execute in parallel for efficiency
- Each produces its own score without waiting for the other

Branch Configuration:

```
branches = [
    {
        "name": "apt_3: amenity scoring",
        "fn": A3,
        "args": (filtered_apartments,),
        "kwargs": {}
    },
    {
        "name": "apt_4: commute scoring",
        "fn": A4,
        "args": (filtered_apartments,),
        "kwargs": {}
    }
]
brn_outs = brn("apt_2", branches)
```

Agent 3

Amenity Scoring (Branch A)

Example Scores:

- River View Lofts: 1.0 (all 3 amenities matched)
- Skyline Tower: 1.0 (all 3 amenities matched)
- Metro Plaza: 0.33 (only gym matched

Input: Filtered apartments + required amenities

Scoring Logic:

```
required = {"parking", "gym", "pets"}
apt_amenities = {"parking", "gym", "pool", "pets"}
matched = required n apt_amenities  # {"parking", "gym", "pets
score = len(matched) / len(required)  # 3/3 = 1.0
```

Agent 4

Commute Scoring (Branch B)

Example Scores:

- Downtown Deluxe: 0.76 (1.2 miles)
- RIver View Lofts: 0.64 (1.8 miles)
- Skyline Tower: 0.50 (2.5 miles)
- Urban Heights: 0.24 (3.8 miles)

Closer apartments = higher score

nput: Filtered apartments + max commute distance (5 miles)

Scoring Logic:

```
max_commute = 5.0 miles
distance = 1.8 miles
score = 1.0 - (distance / max_commute)
score = 1.0 - (1.8 / 5.0) = 0.64
```

Agent 5

Aggregate & Rank (agg)

Input:

- Filtered apartments (from Agent 2)
- Amenity scores (from Agent 3)
- Commute scores (from Agent 4)

Aggregation Formula:

```
final_score = (0.6 × amenity_score) + (0.4 × commute_score)
```

Why 60/40 weighting?

- Amenities are **long-term quality of life**
- Commute matters but is **partially flexible**

Example Calculation:

```
River View Lofts:
  amenity_score = 1.0
  commute_score = 0.64
  final_score = (0.6 × 1.0) + (0.4 × 0.64) = 0.86
```

Operator Used: agg (many-to-one)

Runtime Execution Trace

Console Output:

```
🏢 Apartment Finder Workflow Starting...
============================================================
[seq] apt_1: parse preferences
  ✓ Loaded preferences for user: leo_001
    Budget: $1800
    Location: downtown
    Required amenities: ['parking', 'gym', 'pets']

[seq] apt_2: filter apartments
  ✓ Filtered apartments: 6 matches

[brn] apt_2 -> apt_3: amenity scoring
[brn] apt_2 -> apt_4: commute scoring
  ✓ Amenity scores calculated: 6 apartments
  ✓ Commute scores calculated: 6 apartments

[agg] apt_5: aggregate & rank
  ✓ Recommendations written: apartment_recommendations.json
  ✓ Summary: [summary.txt](http://_vscodecontentref_/4)

🏆 Top Pick: River View Lofts
   $1750/mo | 1 bed | Score: 0.86
   Commute: 1.8 miles

============================================================
✅ Apartment Finder Workflow Complete!
```

Top 3 Apartment Recommendations

01

River View Lofts

Rent: $1750

Score: 0.86

Amenities: All 3

Commute: 1.8 miles

02

Skyline Tower

Rent: $1600

Score: 0.80

Amenities: All 3

Commute: 2.5 miles

03

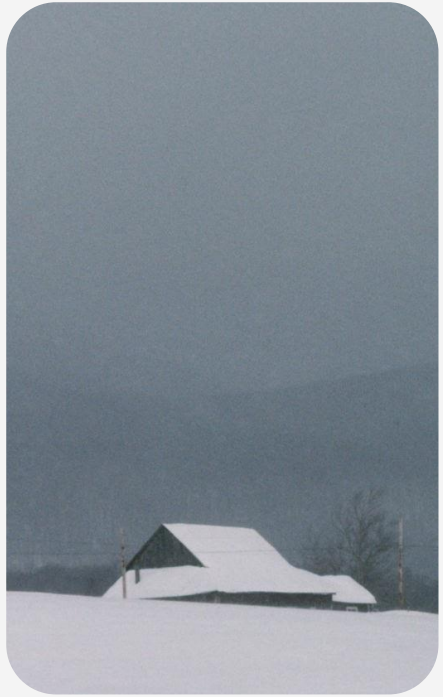
Urban Heights

Rent: $1650

Score: 0.70

Amenities: All 3

Commute: 3.8 miles



Lessons from Workflow Orchestration

01

Modularity Matters

- Each agent has one clear responsibility
- Easy to test, debug, and replace

02

Orchestration Patterns are Powerful

- seq: Simple, linear transformations
- brn: Parallel, independent computations
- agg: Combine results intelligently

03

Contracts Enforce Quality

- Type checking (must return mappings)
- Name uniqueness (no collisions)
- Early error detection

04

Declarative > Imperative

- Define what to compute, not how
- Engine handles execution details
- Easier to reason about

05

Real-World Applications

- Recommendation systems
- Data pipelines
-Multi-stage ML workflows
- Business process automation

Thank you