10 - SequentialCoordinator

Iagro CLP Local — SequentialCoordinator

Runtime **padrão** do CLP Local: desacopla leitura OPC (alta frequência) de tarefas serializadas (DB, fila, captura receita) para reduzir corrida em `write_commands` e carga no PostgreSQL.

Código: `src/core/sequential_coordinator.py`

📋 Resumo

Thread	Função
`read_worker_thread`	Ciclo OPC contínuo → buffer → `on_data_updated`
`serial_worker_thread`	Health, persistência, comandos, receitas

`CLPContainer.monitor_thread` aponta para `serial_worker_thread`.

🏗️ Arquitetura dupla

```
┌──────────────────── read_worker ────────────────────┐
│  _run_read_cycle_once → _detect_changes             │
│  → _value_buffer (lock)                             │
│  → container._on_data_updated → UI + PM (8)         │
└─────────────────────────────────────────────────────┘

┌──────────────────── serial_worker ──────────────────┐
│  loop:                                              │
│    _task_connection_health                          │
│    _task_save_to_database                           │
│    _task_process_commands  → CP (7)                 │
│    _task_monitor_recipes   → RP (5) via RM (6)      │
│  sleep até próximo tick por intervalo               │
└─────────────────────────────────────────────────────┘
```

🔄 Loop serial (`_run_loop`) — tarefas

Ordem	Tarefa	Intervalo típico	Implementação
0	Health OPC	~2000 ms	`_task_connection_health` — detecta queda,

			`mark_connection_lost`
1	Flush buffer leituras	`DB_WRITE_INTERVAL_MS` (~200)	`_task_save_to_database`
2	Fila comandos	`COMMAND_POLL_INTERVAL_MS`	`_task_process_commands`
3	Captura receita	2000 / 10000 adaptativo	`_task_monitor_recipes`

Métricas: `container.metrics.record_serial_loop(duration)`.

`apply_performance_intervals(container.config)` mapeia env → atributos do coordenador na subida do monitor.

🔄 `_task_process_commands` (detalhe)

```
1. check_executing_recipe_parents(container, command_processor)
     └─ sanitize_orphan_create_recipes (fim da função)
2. SE NOT container.is_connected(): RETURN
3. pending ← get_pending_write_commands(limit=10, container_id)
4. PARA cmd IN pending:
     command_processor.execute_command(cmd)
```

Diferença vs CP thread: batch **10** (coordenador) vs **100** (`CommandProcessor.process_commands` em modo thread).

Pré-requisito: `command_processor` existe e **não** está em `run()` paralelo — `_sync_command_processor_thread` garante.

Pós-reconexão

Health worker pode chamar `check_executing_recipe_parents` sem processor após reconnect — reconcilia pais `executing` antes do próximo poll de comandos.

🔄 `_task_monitor_recipes`

Replica gates do **6 - RecipeMonitorService**:

Gate	Fonte
CLP + DB	callbacks / `database_manager`

`has_pending_recipe_commands`	DB
`get_active_recipe`	ApiService
`has_pending_reset`	ProductionMonitor

API: `get_work_order_recipe_sync(status=NA_FILA, limit=1)` — uma candidata por tick.

`recipe_processor.process_recipe(recipe, is_connected)`.

🔄 Leitura OPC (`_read_worker_loop`)

- Prioridades: env `OPCUA_PRIORITY_*` + ids de `get_active_view_variables` (supervisório).
- Mudanças enfileiradas no buffer; serial worker persiste em lote.
- `on_data_updated` dispara na thread de leitura — UI usa scheduler thread-safe.

⚙️ Variáveis de ambiente (via `container.config`)

Chave	Efeito
`DB_WRITE_INTERVAL_MS`	Frequência flush histórico
`COMMAND_POLL_INTERVAL_MS`	Poll fila
`RECIPE_POLLING_FAST_MS` / `SLOW` / `ADAPTIVE_TIMEOUT`	Captura NA_FILA

⚠️ Modo paralelo (contraste)

Aspecto	Com coordenador	Paralelo
`CommandProcessor.run`	**Off**	Thread ativa
`RecipeMonitorService.run`	**Off**	Thread ativa
Leitura	`read_worker`	`VariableMonitorThread` + AutoTuner

Corrida fila	Mitigada (serial + SQL filtro)	Depende de intervalos

🔗 Referências

- **9 - CLPContainer** (`_sync_command_processor_thread`)
- **7 - CommandProcessor** · **6 - RecipeMonitorService**
- **4 - ApiService** (`get_active_view_variables`)