ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Instituto Superior de Engenharia do Porto

Project Report

Técnicas Avançadas de Programação

Prof. Nuno Malheiro
Prof. Nuno Ferreira

21 de junho de 2025

Guilherme Cunha, nº 1201506
José Mendes, nº 1230195
Miguel Domingues, nº 1230200
Pedro Vilarinho, nº 1211149

Index

Milestone 1 – Executive Summary

Problem

The core challenge involves scheduling production orders where each order specifies a quantity of products to manufacture. Each product follows a linear sequence of tasks, with each task requiring specific physical resources and qualified human operators. The system must allocate both physical and human resources efficiently while maintaining production flow.

Domain

The domain is structured around six key entities:

- **Product:** Represents manufacturable items with unique identifiers, names, and ordered task sequences
- **Order:** Defines production requests with quantities and target products
- **Task:** Encapsulates individual production operations with duration and resource requirements
- **PhysicalResource:** Represents factory equipment and machinery by type
- **HumanResource:** Models skilled operators with specific equipment competencies
- **TaskSchedule:** Captures the complete scheduling solution with timing and resource allocation

All entity attributes use opaque types for strong validation and type safety, preventing invalid data through pattern matching (e.g., ProductId must follow "PRD_*" format).

Key Design Decisions

- **Functional Programming:** Immutable data structures and Result[A] type for explicit error handling
- **Strong Typing:** Opaque types prevent runtime errors through compile-time validation
- **Modular Architecture:** Clear separation between domain logic, XML processing, and scheduling algorithms
- **Error Management:** Comprehensive DomainError enumeration replaces exception-based error handling

Algorithm

The scheduling algorithm implemented in Milestone 01 adopts a sequential and deterministic approach based on a First-Come-First-Served (FCFS) strategy. This method prioritizes simplicity and predictability, deliberately avoiding parallel execution or optimization, which are deferred to later milestones.

Step-by-Step Breakdown

1. Data Extraction and Validation

Before scheduling begins, all necessary data is extracted from an input XML file. This includes:

- Orders (each with a product and quantity)
- Products (with an ordered list of task IDs)
- Tasks (with durations and required resource types)
- Human and physical resources

The system uses type-safe wrappers (e.g., **OrderId, ProductNumber, TaskScheduleTime**) to guarantee input integrity, leveraging Scala's functional error-handling model through the **Result** type (an alias for **Either[DomainError, A]**).

2. Sequential Order Processing

Orders are processed one at a time, preserving the sequence found in the input XML. For each order, the associated product is retrieved and for each unit of the product (quantity), the system processes all associated tasks in the exact order specified by the product definition.

3. Task-by-Task Scheduling

Each task is scheduled one after the other using the current global time. Tasks are not allowed to overlap or run concurrently:

- **Resource Allocation:** First, the required physical resources are allocated, ensuring no duplication within the same time slot. Then, qualified human resources are selected, matching the physical resource types.
- **Time Management:** The task is assigned a start time based on the current global time and an end time calculated as start + duration.
- A **TaskSchedule** object is created and prepended to the list of scheduled tasks.

4. Global Time Tracking

A central concept in the algorithm is the global time tracker, passed along each function call and incremented after each task. This ensures:

- Tasks are never scheduled in parallel.
- Resource reuse is implicitly allowed, but only after a task finishes.
- The schedule is sequential not only within a product unit but also across product units and orders.

5. Final Schedule Generation

Once all tasks for all product units of all orders have been scheduled, the resulting list of TaskSchedule entries is transformed into XML and returned.

Milestone 2 – Executive Summary

Generators

1. Basic Type Generators (SimpleTypeGenerator)

The *SimpleTypeGenerator* forms the foundation for all primitive domain types in the system. It includes generators for IDs, such as *ProductId*, *OrderId*, *TaskId*, *PhysicalResourceId*, and *HumanResourceId,* which follow specific prefixes like '*PRD_*', *ORD_*', '*TSK_*', *PRS_*', and 'HRS_', respectively, in line with the XSD specification. Additionally, it provides value generators for *OrderQuantity* (ranging from 1 to 5), *TaskTime* (positive integers), *ProductNumber*, and Human Resources for products and resources.

Each generator follows the same pattern: it first produces a raw candidate (like a random string or number), then validates it through the domain's opaque type constructors (from methods). If validation fails, the generator also fails, ensuring only structurally valid data is created.

2. Complex Entity Generators

Building on the basic types, the following generators create full-fledged domain entities with internal consistency:

a) TaskGenerator

This component generates complete Task instances using valid IDs and durations, along with required *PhysicalResourceTypes* derived from an available list. It supports both generic generation and a **deterministic variant**, which ensures that all required resource types are operable by available human resources. This helps enforce practical constraints during testing.

b) ProductGenerator

Products are generated with valid IDs and names, and each is associated with a non-empty, distinct list of task IDs. This guarantees the product structure references existing tasks and maintains referential integrity.

c) OrderGenerator

Orders are created by linking valid IDs and quantities (1–5) with existing product IDs. This ensures that every order is associated with a real product and carries a valid structure.

d) PhysicalResourceGenerator and HumanResourceGenerator

These generate lists of valid physical and human resources. Physical resources consist of valid IDs and types. Human resources are more involved, they are assigned names, IDs, and a non-empty list of *PhysicalResourceTypes* that represent their operational capabilities. There's also a **deterministic version** that ensures human resources are compatible with the actual types present in the physical resource pool, supporting consistent downstream task assignment.

3. Full Context Generator (*TaskScheduleGenerator*)

At the highest level, the *TaskScheduleGenerator* combines all the individual components to create complete domain datasets. It produces consistent sets of physical resources, human resources with compatible capabilities, tasks that require feasible resource types,

products that use those tasks, and orders linked to those products. It supports both random and deterministic modes. The latter ensures that every generated element is mutually compatible, enabling conflict-free scheduling in simulation or property-based testing environments.

Properties

The properties defined for the scheduling system serve to ensure consistency, validity, and correctness of the generated schedules. They were grouped according to their purpose and relevance to key aspects of scheduling: determinism, resource allocation, data consistency, and execution order. Below is a summary of the main groups of properties:

1. **Validity and Determinism of Schedules**

These properties verify that the schedule produced by the system is both valid and deterministic. This includes ensuring that all tasks required by an order are scheduled in the correct quantity, that each product instance receives all its tasks, and that repeated executions with the same input yield the same output. These checks help confirm that the scheduler behaves predictably and in accordance with the domain's rules.

2. **Resource Allocation Consistency**

Several properties ensure that both human and physical resources are only allocated when they are available and compatible with the task requirements. In addition, they check that no two tasks overlap in time if they share a common resource. These validations are critical to prevent conflicts in resource usage and to guarantee the feasibility of the resulting schedule.

3. **Order Preservation and Task Sequencing**

To respect the domain logic, the scheduler must maintain the sequential order of tasks for each product. Properties in this group ensure that tasks for the same product instance are executed linearly, i.e., in the order defined in the product's task list. This reflects the assumption of a linear production process.

4. **Uniqueness and Identifier Validity**

Another set of properties checks the uniqueness and validity of various identifiers, such as those for orders, products, tasks, and resources. This prevents duplication and ambiguity in schedule entries. It also confirms that generated IDs follow the expected structure and that references (e.g., order referencing a product) are always valid.

5. **Robustness to Resource Ordering and Generator Correctness**

The scheduler should not be sensitive to the order in which resources are provided. Properties in this group verify that shuffling the list of human resources does not affect the outcome. Additionally, they ensure the correctness of the generators used in property-based testing, which is essential for generating valid and representative test data.

Milestone 3 – Executive Summary

Milestone 3 represents the final evolution of the scheduling system, implementing complete parallelization and production order optimization to minimize total production time. Unlike the previous milestones, this allows:

- Simultaneous execution of multiple tasks
- Alteration of product production order
- Parallel production of products from different orders
- Optimization based solely on physical and human resource constraints

Algorithm

1. Main Structure

The **ScheduleMS03** algorithm implements a **greedy approach** with prioritization that operates in temporal batches:

1. Resource validation → Instance creation → Recursive scheduling
2. For each time moment: identify eligible tasks → prioritize → allocate resources
3. Advance time when necessary → repeat until completion

1.1. Applied Heuristics

Task Prioritization Heuristic

```
def prioritizeTasks(tasks: List[TaskInfo]): List[TaskInfo] =
  tasks.sortBy(task =>
    (-task.task.time.to, task.task.physicalResourceTypes.map(rarityMap.getOrElse(_, 0)).sum)
  ), [])
}
```

Prioritization Criteria:

- **Duration (descending):** Longer tasks are prioritized (Longest Processing Time First)
- **Resource rarity:** Tasks using scarcer resources have priority

Justification:

- Long tasks have a greater impact on total time if delayed
- Rare resources should be allocated first to avoid bottlenecks

Optimization Advantages:

- **Makespan Reduction:** By prioritizing long tasks, reduces idle time at the end of production
- **Bottleneck Minimization:** Prioritization by rarity prevents critical resources from being wasted on less important tasks

Resource Allocation Heuristic

```
requiredTypes.sortBy(t =>
  availableResources.count(res => matchesType(res, t) && !usedIds.contains(extractId(res)))
)
```

Strategy:

- Allocates scarcer resources first (lower availability)
- Avoid situations where critical resources become unavailable

Optimization Impact:

- **Deadlock Prevention:** Avoids scenarios where tasks are blocked waiting for resources that will never be released
- **Parallelism Maximization:** Ensures limited resources are used efficiently
- **Wait Time Reduction:** Minimizes time tasks spend in queue waiting for specific resources

1.2. Intelligent Parallelization

- **Temporal Eligibility:**
  - Tasks are only considered if earliestStart ≤ currentTime Products cannot have overlapping tasks (linearity maintained) Dependency verification through productTaskIndex
- **State Management:**
  - The system maintains:
    - Ready Tasks: Tasks ready for execution
    - Resource Availability: Temporal availability of each resource
    - Product Progress: Progress of each individual product
    - Schedules: Complete generated schedule

2. Implemented Optimizations

2.1. Time Advancement

When no task can be scheduled at the current time:

```
val nextResourceTime = state.resourceAvailability.values.filter(_ > minEarliestStart).minOption
```

Calculates the next moment when resources will be available, avoiding unnecessary iterations.

Benefits for Optimal Time:

- **Idle Time Elimination:** Skips periods where no task can be executed, focusing only on productive moments
- **Fast Convergence:** Significantly reduces the number of iterations needed to complete scheduling
- **Temporal Precision**: Ensures the algorithm finds the next viable execution point without computational waste

2.2. Batch Allocation

Maximizes the number of tasks executed simultaneously and avoids resource conflicts through tracking used IDs.

Contribution to Optimization:

- **Throughput Maximization:** Executes the maximum possible number of tasks in parallel at each moment
- **Efficient Utilization:** Makes the most of available physical and human resource capacity
- **Fragmentation Reduction:** Minimizes periods where resources are underutilized

2.3. Prior Validation

```
validateResourceRequirements(tasks, physicalResources, humanResources)
```

Verifies impossibilities before scheduling, avoiding unnecessary processing in unfeasible scenarios.

Heuristics

1. Heuristics Impact Analysis on Optimal Time

To evaluate the impact of different heuristic strategies on production time optimization, we applied each heuristic to the same set of valid production agendas. The results measure how much each solution improves upon the baseline (MS01), expressed as an average percentage of improvement in total makespan.

The two top-performing heuristics were:

- **LJF_RARITY:** Prioritizes longer tasks and rarer resources.
- **WEIGHTED:** Uses a weighted formula combining task duration, resource count, and rarity.

Both heuristics achieved an average improvement of **45.04%**, outperforming all others. This suggests that **prioritizing long tasks with scarce resources** or **applying a**

balanced weighted approach yields the most efficient scheduling results in terms of makespan reduction.

Not far behind, **RESOURCE_EFFICIENCY** (44.67%) and **LEAST_TYPES** (44.59%) also performed competitively, indicating that strategies focusing on resource utilization can also be effective.

Here is the full ranking of heuristics by average improvement:

Heuristic	Description	Avg. Improvement
LJF_RARITY	Longest jobs first + resource rarity	45.04%
WEIGHTED	Weighted: (2x duration + resource count + rarity)	45.04%
RESOURCE_EFFICIENCY	Efficiency: resources per unit of time	44.67%
LEAST_TYPES	Minimize diversity of required resource types	44.59%
RARITY_FIRST	Rarest resources first + longest jobs	44.17%
SJF_RARITY	Shortest jobs first + resource rarity	43.54%
FIFO	First-In-First-Out with rarity as tiebreaker	43.50%
BALANCED	40% duration + 40% rarity + 20% original order	43.04%

Alternative algorithms

During the development of the scheduling system, several alternative algorithmic approaches were considered and evaluated for their applicability to the production scheduling domain. While the implemented greedy approach with intelligent heuristics proved most suitable for our specific constraints, it is valuable to analyze why other well-established algorithms were not adopted.

- **Brute Force Algorithm**: A brute force approach would generate all possible task orderings and resource assignments, evaluating each combination for feasibility and optimality. While this guarantees the finding of the optimal solution, it is computationally intractable for real-world manufacturing scenarios due to the

exponential growth in combinations with the number of tasks, resources, and orders.

- **Greedy Algorithm**: A pure greedy approach would select tasks based on simple heuristics (e.g., shortest processing time first, earliest due date) without considering future implications. While computationally efficient, this approach often leads to suboptimal solutions in complex manufacturing environments where resource contention and task dependencies create intricate interdependencies that require more sophisticated planning.

- **Critical Path Method (CPM)**: CPM focuses on identifying the longest sequence of dependent tasks to minimize project completion time. However, our manufacturing scheduling problem involves multiple parallel product instances with shared resources, making traditional CPM insufficient as it doesn't handle resource constraints effectively.

- **Resource-Constrained Project Scheduling (RCPS)**: Classical RCPS algorithms like priority-rule based heuristics could be applied, but they typically assume unlimited resource types and don't handle the specific constraints of our domain, such as human resources with multiple skill sets and the need for simultaneous physical and human resource allocation.

- **Constraint Satisfaction Problem (CSP)**: The scheduling problem could be modeled as a CSP with variables representing task start times and resource assignments. However, the dynamic nature of resource availability and the need for efficient incremental scheduling make direct CSP solving less suitable for our real-time scheduling requirements.

- **Hungarian Algorithm**: The Hungarian Algorithm is used to find maximum weighted matching in bipartite graphs, which could theoretically be applied to match tasks with time slots. However, our problem requires complex resource allocation where multiple resources of different types must be assigned simultaneously to each task, making the one-to-one matching limitation of Hungarian Algorithm inadequate for our multi-resource constraints.