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.