Requirements Analysis

v1.0

Warehouse Robot Management System

Contents

1 Changelog

- v1.0: Original version.

2 Requirements Analysis

This comprehensive warehouse automation system must coordinate multiple robots working collaboratively to manage warehouse operations through specialized robot classifications and integrated workflow management.

2.1 Functional Requirements

2.1.1 Robot Management Requirements

The system must provide comprehensive robot lifecycle management including creation, classification, task coordination, and operational monitoring capabilities.

- **Robot Instance Management**
  - **Robot Creation**: Create robot instances with unique identifiers (`robot_id_`), model specifications (`model_`), and operational status tracking (`operational_status_`).
  - **Robot Classification**: Support three distinct robot types with specialized capabilities: Carrier, Scanner, and Sorter robots implementing polymorphic behavior.
  - **Mobility Functions**: Provide fundamental navigation capabilities through `move()` method for all robot types to traverse warehouse facility.
  - **Status Tracking**: Monitor robot operational status through `get_status()` method and maintain current position data.
- **Task Assignment System**
  - **Polymorphic Task Execution**: Execute specialized tasks through abstract `execute_task()` method implemented differently by each robot type.
  - **Task Description Interface**: Provide task identification through abstract `get_task_description()` method returning type-specific operation descriptions.
  - **Operator Assignment**: Support operator assignment through `assign_operator()` method for robot control coordination.

2.1.2 Battery Management Requirements

The system must implement robust power management functionality to monitor battery status, coordinate charging operations, and maintain optimal energy distribution across all robotic units.

- **Power Monitoring System**
  - **Capacity Management**: Monitor battery capacity specifications (`capacity_`) and charging status (`charging_status_`) for each power unit.
  - **Charge Level Monitoring**: Track current charge levels (`charge_level_`) through `get_charge_level()` method in real-time during robot operations.
  - **Battery Status Detection**: Implement `is_low_battery()` method to detect critical power levels requiring immediate charging intervention.
- **Charging Management System**
  - **Charging Operations**: Provide battery recharging functionality through `charge()` method when charge levels become insufficient.
  - **Discharge Management**: Control power consumption through `discharge()` method during robot operations.
  - **Time Estimation**: Calculate remaining operational time through `get_remaining_time()` method for charging schedule coordination.
- **Battery Assignment System**

- **Composition Relationship**: Maintain strong ownership relationship where each robot owns exactly one battery instance.
- **Battery Integration**: Support battery assignment through `assign_battery()` method with unique battery identification (`battery_id_`).

2.1.3 Operator Management Requirements

The system must support comprehensive operator management including profile administration, authentication, and robot control interfaces to enable effective human oversight of warehouse operations.

- **Operator Registration System**
  - **Profile Management**: Register and maintain operator profiles with unique identification (`name_`) and operator credentials (`operator_id_`).
  - **Session Management**: Track operator login sessions through `active_session_` attribute and provide `login()` and `logout()` methods.
  - **Authentication System**: Implement credential validation through `validate_credentials()` method for secure system access.
- **Robot Control System**
  - **Task Assignment Interface**: Enable operators to assign tasks through `assign_task()` method to designated robots.
  - **Multi-Robot Management**: Support operators managing multiple robots simultaneously through aggregation relationship.
  - **Monitoring Capabilities**: Provide robot status monitoring through `monitor_robots()` method for operational oversight.
  - **Emergency Control**: Implement `emergency_stop()` method for immediate cessation of all robot operations.

2.1.4 Specialized Robot Function Requirements

- **Carrier Robot Operations**
  - **Load Management**: Monitor load capacity (`load_capacity_`) and current load (`current_load_`) for safe transportation operations.
  - **Item Handling**: Provide `load_item()` and `unload_item()` methods for cargo manipulation with weight validation.
  - **Route Planning**: Implement `calculate_route()` method returning optimized position sequences for efficient navigation.
  - **Safety Validation**: Include `check_weight_limits()` method to ensure safe operational parameters.
- **Scanner Robot Operations**
  - **Scanning Capabilities**: Implement `scan_barcode()` method with configurable scanner range (`scanner_range_`) and accuracy (`scan_accuracy_`).
  - **Inventory Management**: Provide `update_inventory()` method for real-time inventory record updates.
  - **Data Validation**: Include `validate_scan()` method to ensure scan accuracy and data integrity.
  - **Database Synchronization**: Implement `sync_database()` method for inventory system integration.
- **Sorter Robot Operations**
  - **Item Categorization**: Implement `categorize_item()` method with configurable sorting accuracy (`sort_accuracy_`).
  - **Sorting Logic**: Provide `apply_sorting_rules()` method for consistent categorization processes.
  - **Zone Management**: Include `move_to_zone()` method for proper item placement in designated warehouse areas.
  - **Performance Tracking**: Implement `update_sort_statistics()` method for operational metrics collection.

2.1.5 Supporting System Requirements

- **Position Management**

- **Coordinate Tracking**: Maintain robot positions with coordinates (`x_`, `y_`), zone information (`zone_`), and timestamps (`timestamp_`).
- **Distance Calculations**: Provide `calculate_distance()` method for navigation and route optimization.
- **Position Validation**: Implement `is_valid_position()` method to ensure robots remain within operational boundaries.
- **Coordinate Updates**: Support position updates through `update_coordinates()` method for real-time tracking.

- **Task Management**
  - **Task Identification**: Maintain task records with unique identifiers (`task_id_`), types (`task_type_`), priorities (`priority_`), and status (`status_`).
  - **Robot Assignment**: Provide `assign_to_robot()` method for task distribution to appropriate robot types.
  - **Status Management**: Implement `update_status()` method for task lifecycle tracking.
  - **Time Estimation**: Include `calculate_completion_time()` method for scheduling and resource planning.
- **Item Management**
  - **Item Tracking**: Maintain item records with identification (`item_id_`), physical properties (`weight_`), categorization (`category_`), and location (`location_`).
  - **Property Access**: Provide `get_weight()` and `get_category()` methods for item processing decisions.
  - **Location Management**: Implement `set_location()` method for inventory tracking and warehouse organization.
- **Database Management**
  - **Inventory Data Storage**: Maintain persistent storage of inventory records through `inventory_records_` mapping system for efficient data access.
  - **Database Synchronization**: Provide `sync_database()` method for coordinating inventory updates from scanning robots with central database systems.
  - **Data Validation**: Implement `validate_inventory_changes()` method to ensure data integrity before committing updates to persistent storage.
  - **Connection Management**: Support `database_connection_` establishment and maintenance for reliable data persistence operations.
  - **Audit Trail**: Include `log_database_operation()` capability for comprehensive tracking of all database modifications and access patterns.

2.2 Non-Functional Requirements

2.2.1 Performance Requirements

- **Concurrent Operations**: Handle simultaneous operations from multiple robots and operators without performance degradation.
- **Response Time**: Ensure robot response time to operator commands remains within acceptable operational thresholds.
- **Real-time Monitoring**: Provide real-time battery monitoring and status updates to prevent operational disruptions.
- **Throughput Management**: Maintain warehouse throughput efficiency during peak operational periods.
- **Method Execution Efficiency**: Ensure polymorphic method calls (`execute_task()`, `get_task_description()`) execute efficiently across all robot types.

2.2.2 Reliability Requirements

- **System Continuity**: Maintain operational continuity when individual robots require maintenance or charging.
- **Fault Tolerance**: Prevent battery depletion from causing system-wide operational failures.
- **State Management**: Ensure operator disconnections do not leave robots in undefined or unsafe operational states.
- **Error Recovery**: Implement automatic recovery procedures for common operational failures.
- **Data Integrity**: Maintain consistency of robot positions, battery levels, task assignments, and inventory records across system operations.

- **Database Reliability**: Ensure reliable database operations with proper connection management and transaction handling for inventory data.

2.2.3 Scalability Requirements

- **System Expansion**: Accommodate additional robots, operators, and battery units as warehouse operations expand.
- **Robot Type Integration**: Support integration of new robot classifications without requiring complete system redesign through inheritance mechanisms.
- **Load Distribution**: Distribute operational loads efficiently across available robot resources.
- **Polymorphic Extension**: Enable addition of new robot types by extending the abstract Robot class with specialized implementations.

2.2.4 Safety Requirements

- **Collision Avoidance**: Implement collision detection and avoidance mechanisms for all robot movements using position tracking.
- **Emergency Protocols**: Provide emergency stop functionality accessible to all operators through `emergency_stop()` method.
- **Battery Safety**: Include safety protocols for battery charging operations to prevent overcharging or thermal damage.
- **Operational Boundaries**: Enforce warehouse operational boundaries through `is_valid_position()` validation method.
- **Weight Safety**: Ensure load safety through `check_weight_limits()` method in carrier robot operations.

2.3 Technical Constraints

2.3.1 System Architecture Constraints

- **Robot-Battery Relationship**: Each robot must maintain exactly one battery assignment through composition relationship at any given operational time.
- **Battery Dependency**: Robot operations are strictly dependent on sufficient battery charge levels for continued functionality.
- **Operator Assignment**: Robot-operator assignments must be clearly defined and properly enforced through aggregation relationship throughout system operations.
- **Abstract Method Implementation**: All concrete robot classes must implement abstract methods `execute_task()` and `get_task_description()`.
- **Communication Protocols**: Establish reliable communication channels between robots, operators, and central management system.

2.3.2 Object-Oriented Design Constraints

- **Inheritance Hierarchy**: All specialized robots must inherit from the abstract Robot base class maintaining polymorphic behavior.
- **Composition Enforcement**: Robot instances must own their Battery and Position objects with strong lifecycle dependency.
- **Aggregation Relationships**: Operators must manage robots through weak ownership allowing independent robot existence.
- **Encapsulation Requirements**: Private attributes must be accessed only through public method interfaces maintaining data integrity.
- **Visibility Modifiers**: Maintain proper access control with private (-), protected (#), and public (+) visibility as specified in class design.

2.3.3 Integration Constraints

- **Legacy System Integration**: System must integrate seamlessly with existing warehouse management infrastructure.
- **Data Compatibility**: Maintain compatibility with current inventory management and tracking systems through standardized database interfaces.
- **Database Integration**: Ensure seamless integration with existing inventory database systems and data formats.
- **Hardware Limitations**: Work within existing warehouse physical constraints and equipment limitations.
- **Enumeration Consistency**: Maintain consistent status values through defined enumerations (RobotStatus, ChargingStatus, TaskType, Priority, TaskStatus).

2.3.4 Operational Constraints

- **Charging Downtime**: Minimize robot downtime during battery charging operations to maintain warehouse efficiency.
- **Maintenance Windows**: Coordinate maintenance schedules to prevent operational disruptions.
- **Audit Requirements**: Provide comprehensive audit trails for all robot activities, operator commands, and database operations.
- **Database Performance**: Minimize performance impact of database synchronization operations during high-frequency scanning activities.
- **Method Call Overhead**: Minimize performance impact of polymorphic method dispatch in high-frequency operations.

2.4 Success Criteria

- **Multi-Robot Coordination**: Successful coordination of multiple robots performing simultaneous warehouse operations without conflicts.
- **Effective Battery Management**: Comprehensive battery management system preventing operational interruptions due to power depletion.
- **Operator Control Efficiency**: Clear and responsive operator control interface enabling effective robot management and task assignment.
- **Specialized Function Accuracy**: Accurate execution of specialized functions by each robot type meeting operational requirements and quality standards.
- **System Integration Success**: Seamless integration of all system components creating a cohesive warehouse automation solution.
- **Operational Efficiency**: Measurable improvement in warehouse operational efficiency and productivity metrics.
- **Safety Compliance**: Full compliance with safety protocols and emergency response procedures.
- **Scalability Demonstration**: Proven ability to expand system capacity without compromising performance or reliability.
- **Polymorphic Behavior Validation**: Successful demonstration of different robot types executing specialized tasks through common interface.
- **Relationship Integrity**: Proper maintenance of composition, aggregation, and inheritance relationships throughout system lifecycle.
- **Database Integration Success**: Reliable inventory data synchronization between scanning robots and central database systems with minimal data loss or corruption.