Multi-Agent System for Automated Skateboard Assembly Plant

Group 4: Irfan Ullah, Sajid - MAC 2025
University Jean Monnet, Ecole de Mines de Saint-Étienne
Course: Cyber Physical and Social Systems: AI & IoT

Abstract

This report presents the design and implementation of a multi-agent system (MAS) for managing a lean automated skateboard assembly plant using the JaCaMo platform. The system coordinates four distinct agent types: Customer, Assembly, Supply, and Workstation agents, to process customer orders through a complete manufacturing pipeline. The implementation demonstrates sophisticated multi-agent coordination through four operational phases: order intake via tuple space communication, supplier contracting using the FIPA Contract Net Protocol, workstation allocation with energy optimization, and sequential assembly execution with real-time visualization.

The system employs CArtAgO artifacts to model the manufacturing environment, enabling shared resources with mutual exclusion guarantees for workstation access. Three supply agents implement distinct bidding strategies, fast delivery, cost optimization, and quality focus, competing through auctions to fulfill part requirements. The assembly process supports configurable skateboard specifications including optional components such as protective rails and IoT connectivity modules. A graphical user interface provides real-time progress tracking for each order, displaying assembly stages, parts checklists, and completion status. The project successfully demonstrates the integration of agent-oriented programming, environment modeling, and interaction protocols within a practical manufacturing scenario, achieving concurrent processing of multiple orders while respecting customer constraints on cost, delivery time, and energy consumption.

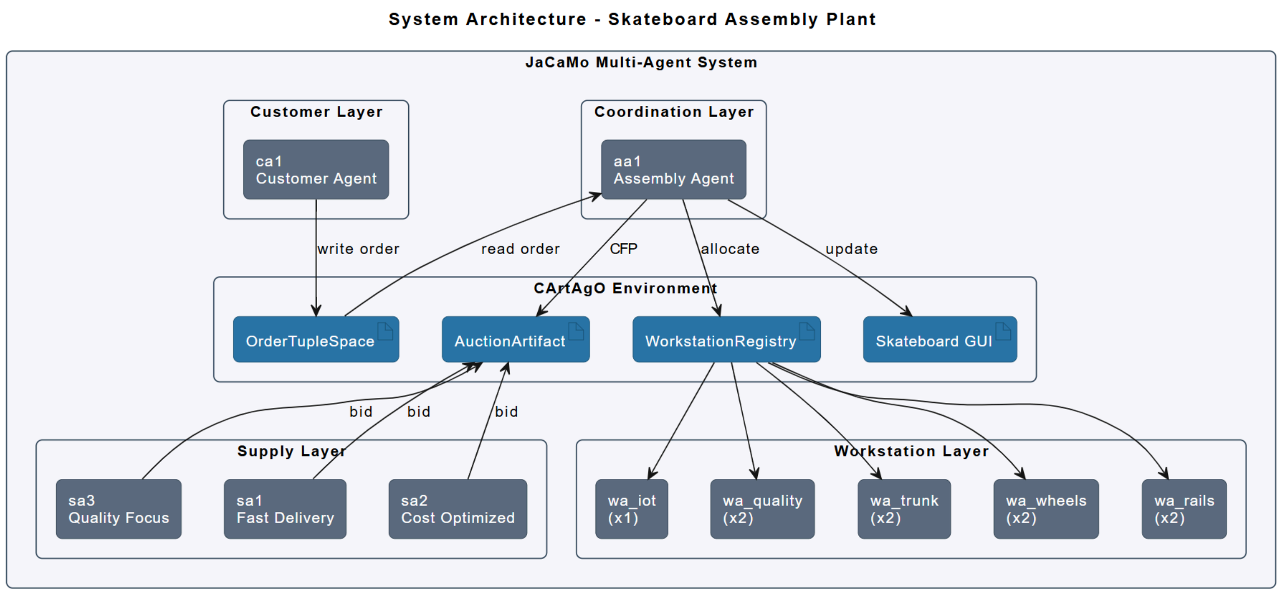
1. Overview

1.1 System Architecture

The skateboard assembly plant is modeled as a multi-agent system comprising fourteen autonomous agents organized into four functional categories. The Customer Agent represents end-users who submit skateboard orders with specific configurations and constraints. The Assembly Agent serves as the central coordinator, managing the entire manufacturing process from order receipt to final delivery. Three Supply Agents compete to provide parts through auction mechanisms, each implementing a distinct business strategy. Ten Workstation Agents control physical assembly operations, with two instances for each workstation type to enable parallel processing.

The system architecture follows the JaCaMo paradigm, integrating three complementary dimensions: Jason for BDI (Belief-Desire-Intention) agent programming, CArtAgO for environment artifact modeling, and

Moise for organizational structure definition. This separation of concerns enables clean modularization where agent reasoning is decoupled from environmental interactions and organizational constraints.



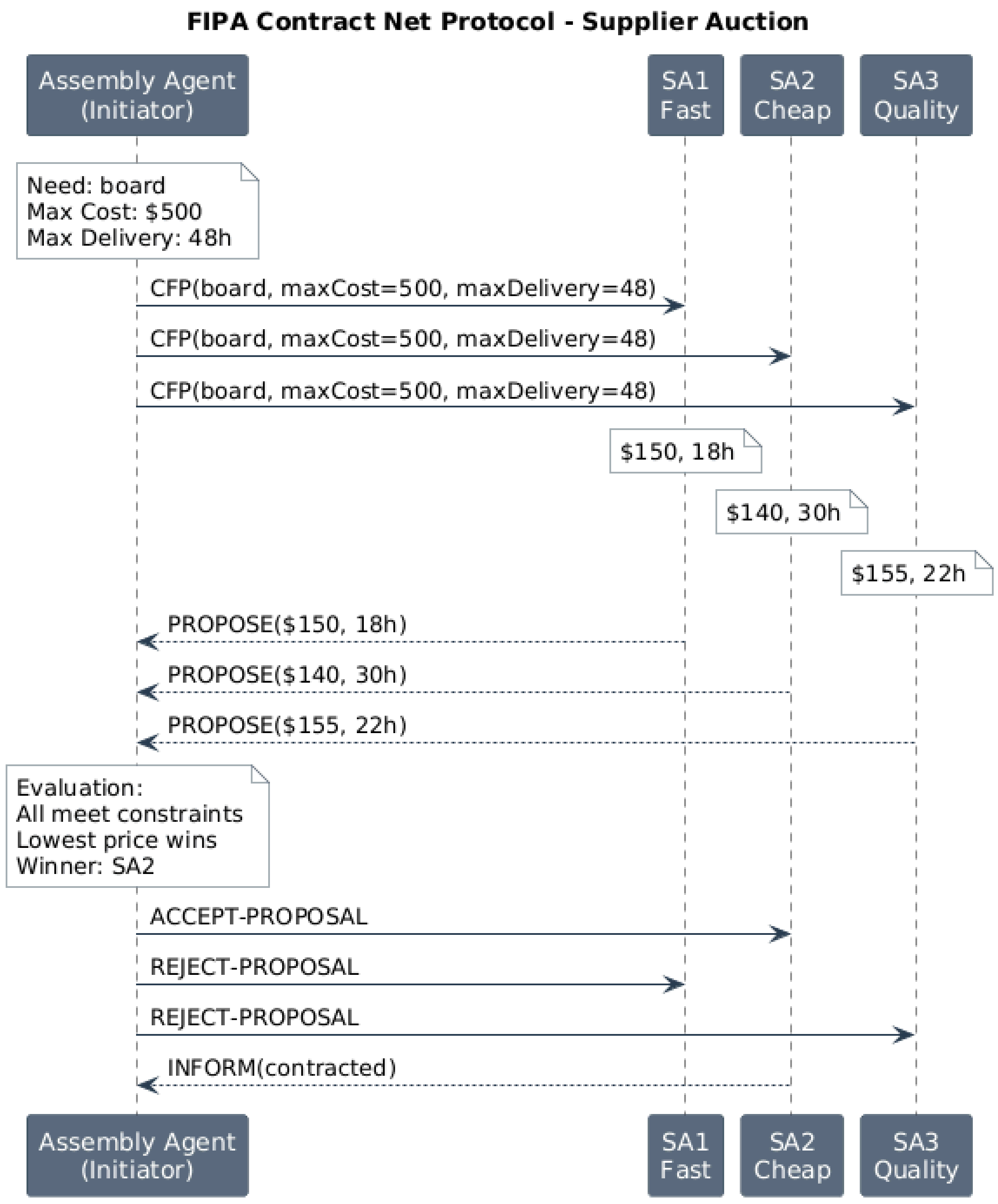
1.2 Coordination Mechanisms

The multi-agent system employs four complementary coordination approaches as defined in the JaCaMo framework:

Agent-Centered Coordination: Each agent maintains its own belief base containing knowledge about capabilities, ongoing commitments, and system state. The Assembly Agent stores order details, selected suppliers, and allocated workstations as beliefs, enabling autonomous decision-making based on local knowledge. Supply Agents maintain beliefs about their pricing, delivery times, and reputation scores, which inform bidding decisions.

Environment-Centered Coordination: CArtAgO artifacts provide shared environmental resources that multiple agents can perceive and act upon. The OrderTupleSpace artifact implements Linda-style tuple space communication for decoupled order submission. The WorkstationRegistry artifact maintains global state about workstation availability, enabling coordination without direct agent-to-agent communication. Workstation artifacts enforce mutual exclusion through their availability status, preventing concurrent access conflicts.

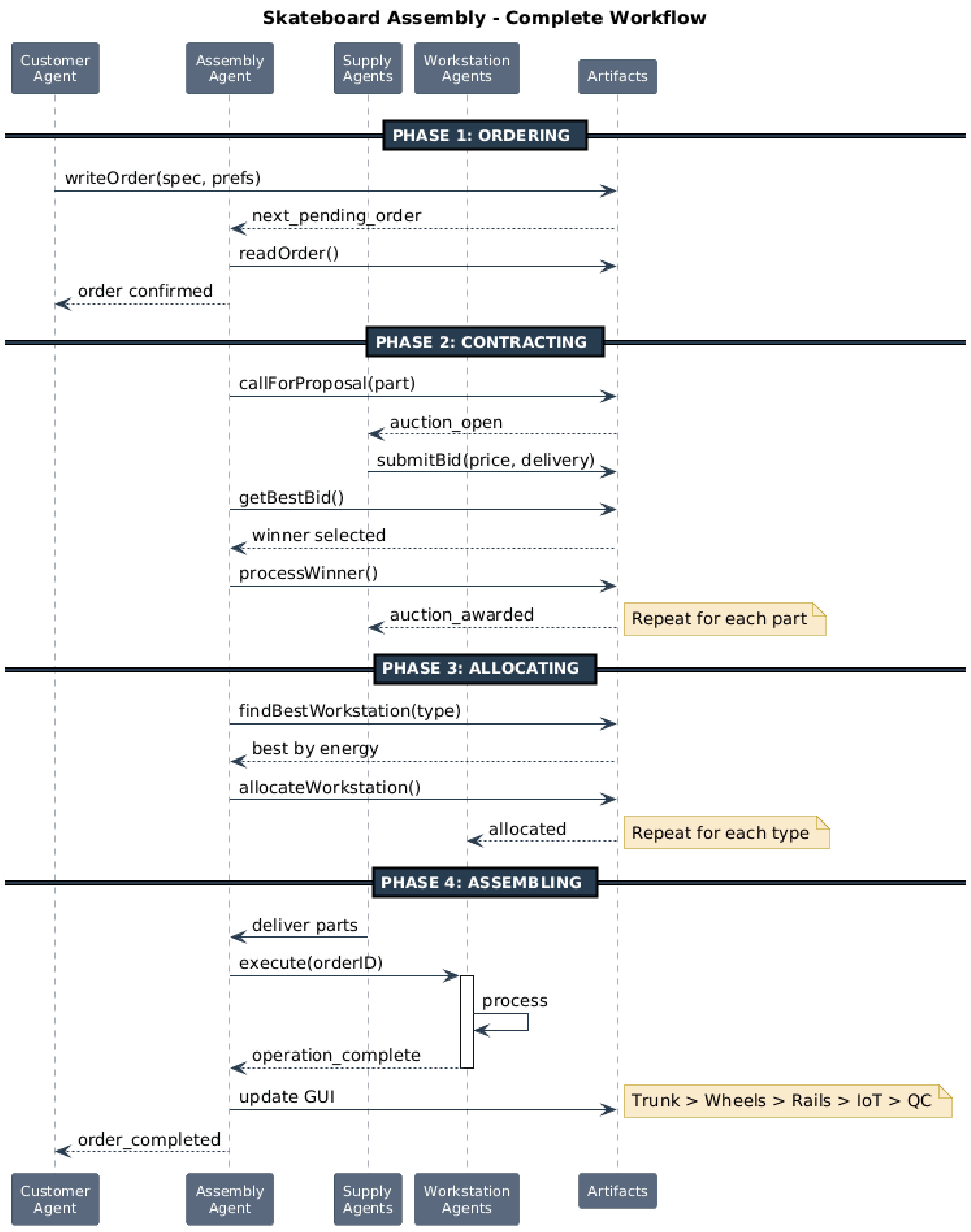
Interaction-Centered Coordination: The FIPA Contract Net Protocol governs supplier selection through a structured auction mechanism. The Assembly Agent issues Calls for Proposal (CFP) specifying part requirements and constraints. Supply Agents evaluate CFPs against their capabilities and submit bids. The Assembly Agent evaluates all bids against cost and delivery criteria, awarding contracts to winning suppliers and notifying rejected bidders.



Organization-Centered Coordination: A Moise organizational specification defines roles, groups, and normative obligations for the assembly process. Roles include customer, assemblyManager, various supplier specializations, and workstation-specific assembler roles. The functional specification defines goal decomposition with sequential and parallel operators, establishing the required ordering of assembly operations. Norms obligate role-holders to commit to specific missions, ensuring accountability for goal achievement.

1.3 Operational Phases

The manufacturing process proceeds through four sequential phases. In the Ordering Phase, customers submit skateboard configurations specifying board type, component quantities, and optional features (rails and connectivity), along with preference constraints for maximum acceptable cost, delivery time, and energy consumption. The Contracting Phase conducts parallel auctions for each required part type, with supply agents competing based on their strategic positioning. The Allocating Phase selects workstations for each assembly operation, optimizing for minimal energy consumption while respecting availability constraints. Finally, the Assembling Phase coordinates part delivery and sequential workstation execution, culminating in quality control verification.



2. Implementation

2.1 Agent Design

All agents are implemented in Jason's AgentSpeak language, following the BDI architecture. The Customer Agent employs dynamic order discovery using the .findall internal action to automatically process all defined order specifications without requiring manual plan modifications. Order specifications are stored as initial beliefs, and a recursive plan iterates through the order list, submitting each to the tuple space.

The Assembly Agent implements the central coordination logic through event-driven plans. Upon detecting a new pending order via observable properties from the OrderTupleSpace, it triggers the contracting phase. The agent maintains beliefs about order details, selected suppliers, and workstation allocations, enabling context-sensitive plan selection. Critical implementation decisions include using artifact operations rather than direct messaging for environment interaction, and employing observable property signals for asynchronous event notification.

Supply Agents implement three distinct bidding strategies through identical plan structures but different initial beliefs. SA1 (Fast Delivery) offers higher prices with shorter delivery times, SA2 (Cost Optimization) provides the lowest prices with longer delivery windows, and SA3 (Quality Focus) balances mid-range pricing with highest reputation scores. Each agent monitors the auction artifact for CFP announcements, evaluates whether it can supply the requested part, and submits bids containing price and delivery time. The term-to-string conversion handles the dynamic matching between auction part types and agent capability beliefs.

Workstation Agents register themselves with the WorkstationRegistry upon initialization, declaring their type, energy consumption, and execution time. They respond to execution requests by locking their associated workstation artifact, simulating the assembly operation through a timed delay, and signaling completion to the Assembly Agent.

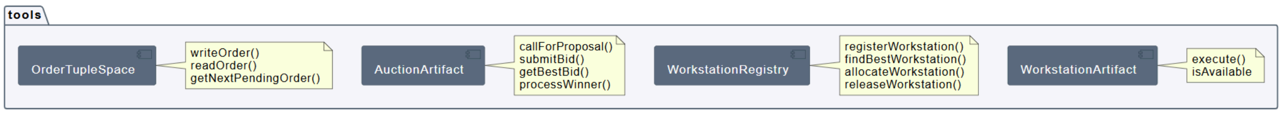
2.2 Artifact Implementation

The OrderTupleSpace artifact implements a simplified Linda tuple space for Customer-Assembly communication. Orders are stored in a HashMap with order ID as key, containing customer ID, specification string, preferences string, and status. The artifact exposes operations for writing orders, reading order data, and updating status. Observable properties track pending and confirmed order counts, while the getNextPendingOrder operation enables polling for new orders.

The AuctionArtifact implements the FIPA Contract Net Protocol mechanics. The callForProposal operation creates a new auction entry with unique identifier, part type, quantity, and constraint parameters. Supply agents invoke submitBid to register their proposals, which are stored in a per-auction list. The getBestBid operation implements the winner selection algorithm, iterating through all bids to find the lowest-price offer that satisfies both cost and delivery constraints. Observable properties signal auction lifecycle events (open, awarded, closed) that agents perceive to coordinate their participation.

The WorkstationRegistryArtifact maintains a central registry of all workstation agents and their current allocation status. The findBestWorkstation operation queries available workstations of a specified type and returns the one with minimum energy consumption. Allocation and release operations update the registry state, with the allocatedTo field tracking which order currently holds each workstation. This centralized registry pattern simplifies the allocation logic compared to distributed negotiation while still enabling energy-optimized selection.

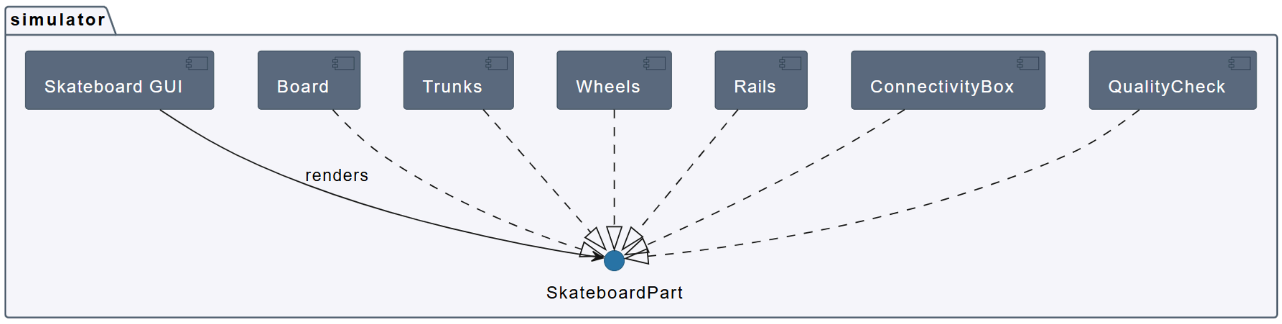
Individual WorkstationArtifact instances model physical workstations with execute operations that simulate assembly delays. The isAvailable observable property implements mutual exclusion, when a workstation begins execution, availability is set to false, preventing other orders from allocation until the operation completes.



2.3 GUI Visualization

The Skateboard GUI artifact extends CArtAgO's GUIArtifact to provide real-time assembly visualization. Each order receives its own dedicated window instance, solving the multi-order display overlap problem inherent in shared GUI approaches. The visualization includes a progress bar tracking completion through six stages (Board, Trunks, Wheels, Rails, IoT, Quality Control), an assembly area rendering skateboard components with detailed graphics, a parts checklist tracking completed items by name and a status bar displaying order information and completion state.

Each skateboard component is implemented as a separate Java class (Board, Trunks, Wheels, Rails, ConnectivityBox, QualityCheck) implementing the SkateboardPart interface. Components render with professional styling including gradients, shadows, and detailed visual elements. The part-by-name tracking ensures correct checklist behavior regardless of which optional components are included in a particular order configuration.



2.4 Decision Mechanisms

Supplier selection employs a constraint-satisfaction approach where bids must meet both maximum cost and maximum delivery time requirements specified in customer preferences. Among qualifying bids, the lowest-price bid wins, implementing a cost-minimization strategy. This approach consistently selects SA2

(Cost Optimization) when delivery constraints are not tight, demonstrating the expected behavior of the competitive auction mechanism.

Workstation allocation follows an energy-minimization strategy. The registry queries all available workstations of the required type and selects the one with lowest energy consumption. This greedy approach provides reasonable energy efficiency while maintaining simplicity. The allocation sequence follows the assembly dependency order: trunks must be mounted before wheels, and quality control must be last.

3. Execution Instructions

3.1 Prerequisites

The system requires Java JDK 17 or higher, Gradle 7.0 or higher, and JaCaMo 1.2 or higher. Ensure that JAVA_HOME environment variable points to your JDK installation and that Gradle is available in your system PATH.

3.2 Project Setup

Clone the repository from GitLab using the command: git clone https://gitlab.emse.fr/mac2025/group4.git. Navigate to the project directory with cd group4. The project follows standard JaCaMo structure with agent source files in src/agt/, Java artifacts in src/env/, and the project configuration in mASB.jcm.

3.3 Building and Running

Build the project using ./gradlew build on Linux/Mac or gradlew.bat build on Windows. Run the system with ./gradlew run. The system will initialize all agents and artifacts, then begin processing orders automatically after a 5-second startup delay. GUI windows will appear for each order as assembly begins.

3.4 Configuration

To modify orders, edit src/agt/customer_agent.asl. Add new orders by defining order_spec and order_prefs beliefs with unique order IDs. The specification format is:

specification(board_type, trunk_count, wheel_count, rails_yes_no, connectivity_yes_no).
The preferences format is:
preferences(max_cost, max_delivery_hours, max_energy).
Orders are automatically discovered and processed without requiring plan modifications.

```
// Order specification: board_type, trunks, wheels, rails(yes/no), connectivity(yes/no)
order_spec(ord001, "specification(maple_deck, 2, 4, no, no)").    // Basic
order_spec(ord002, "specification(bamboo_deck, 2, 4, yes, no)").  // With rails
order_spec(ord003, "specification(maple_deck, 2, 4, yes, yes)").  // Full
```

```
// Order preferences: max_cost, max_delivery_hours, max_energy
order_prefs(ord001, "preferences(500, 48, 100)").
order_prefs(ord002, "preferences(700, 72, 150)").
order_prefs(ord003, "preferences(1000, 96, 200)").
```

Workstation configurations can be modified in mASB.jcm by adjusting the initial beliefs for energy_consumption and execution_time parameters. Supply agent pricing and delivery times can be modified in their respective .asl files by editing the can_supply beliefs.

4. Discussion

4.1 Strengths

The implemented system demonstrates several notable strengths. The modular architecture cleanly separates agent reasoning, environment modeling, and coordination protocols, enabling independent modification of each component. The artifact-based environment abstraction provides a realistic model of manufacturing resources while enabling coordination through shared state rather than complex messaging protocols.

The FIPA Contract Net implementation successfully creates competitive dynamics among supply agents, with observable cost savings compared to fixed-price procurement. The energy-optimized workstation allocation demonstrates constraint-based decision making while maintaining straightforward implementation. The concurrent order processing capability, enabled by mutual exclusion on workstation resources, allows multiple skateboards to be manufactured simultaneously with proper resource sharing.

The dynamic order discovery mechanism using .findall eliminates the need for manual plan updates when adding orders, improving maintainability. The per-order GUI visualization provides clear feedback on system operation and successfully handles multiple concurrent assemblies without display conflicts.

4.2 Limitations

Several limitations warrant discussion. The winner selection algorithm considers only cost and delivery time, ignoring the reputation scores maintained by supply agents. Incorporating reputation into a weighted multi-criteria evaluation would better reflect real-world procurement decisions. The current greedy workstation allocation does not consider global optimization across multiple concurrent orders, potentially leading to suboptimal overall energy consumption.

The system lacks failure handling mechanisms. If a supply agent fails to deliver or a workstation malfunctions, the assembly would stall indefinitely. Production systems would require timeout mechanisms, alternative supplier selection, and graceful degradation strategies. The Moise organizational specification, while defined, is not actively enforced at runtime, agents operate through direct coordination rather than organizational norm compliance.

4.3 Possible Improvements

Several enhancements could improve the system. Implementing dynamic reputation updates based on delivery performance would create adaptive supplier selection. Adding support for partial order fulfillment and re-auctioning when no suppliers meet constraints would improve robustness. Integrating the Moise organization at runtime through ORA4MAS would enable norm-based coordination with violation detection.

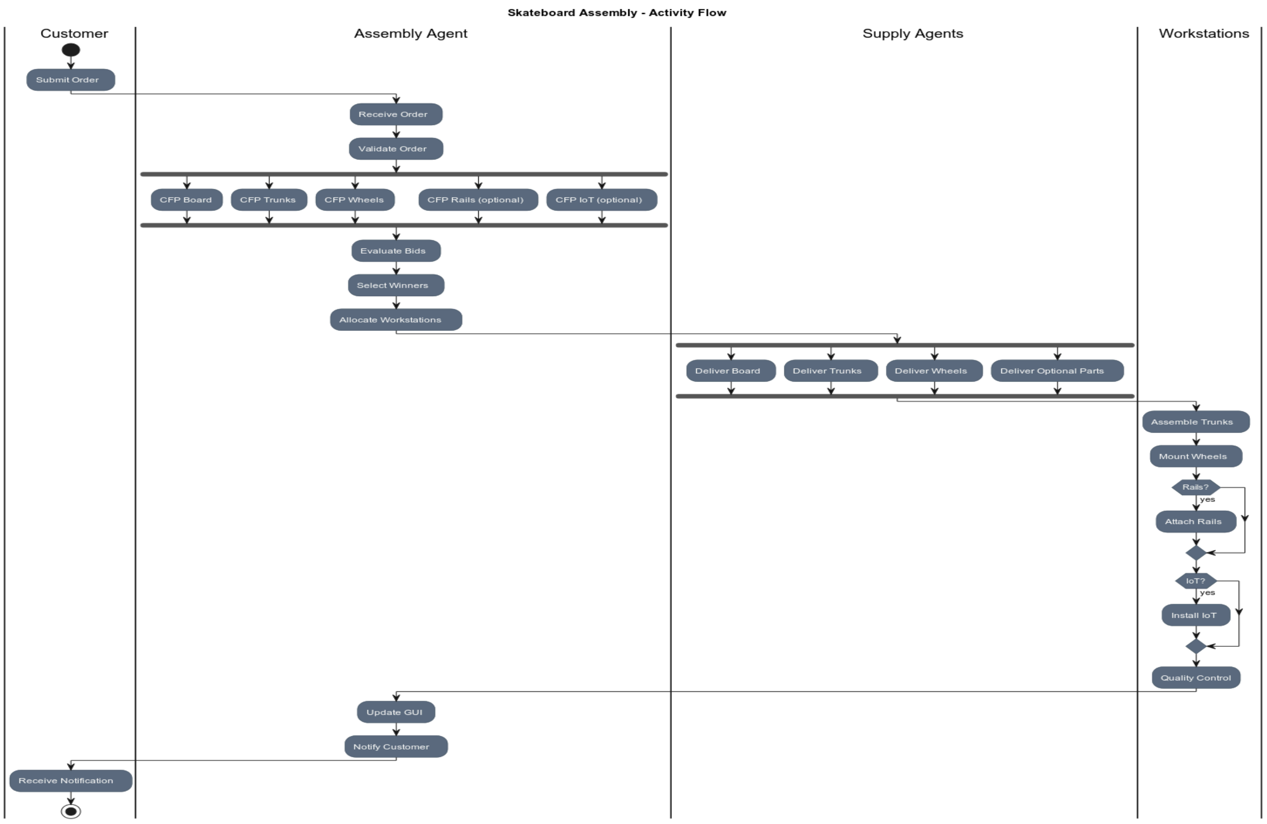
A global workstation scheduler considering all pending orders could optimize overall plant efficiency. Supporting multiple customer agents with priority-based order scheduling would demonstrate more realistic production scenarios. Finally, adding a web-based monitoring dashboard would improve system observability for demonstration and debugging purposes.

4.4 Conclusion

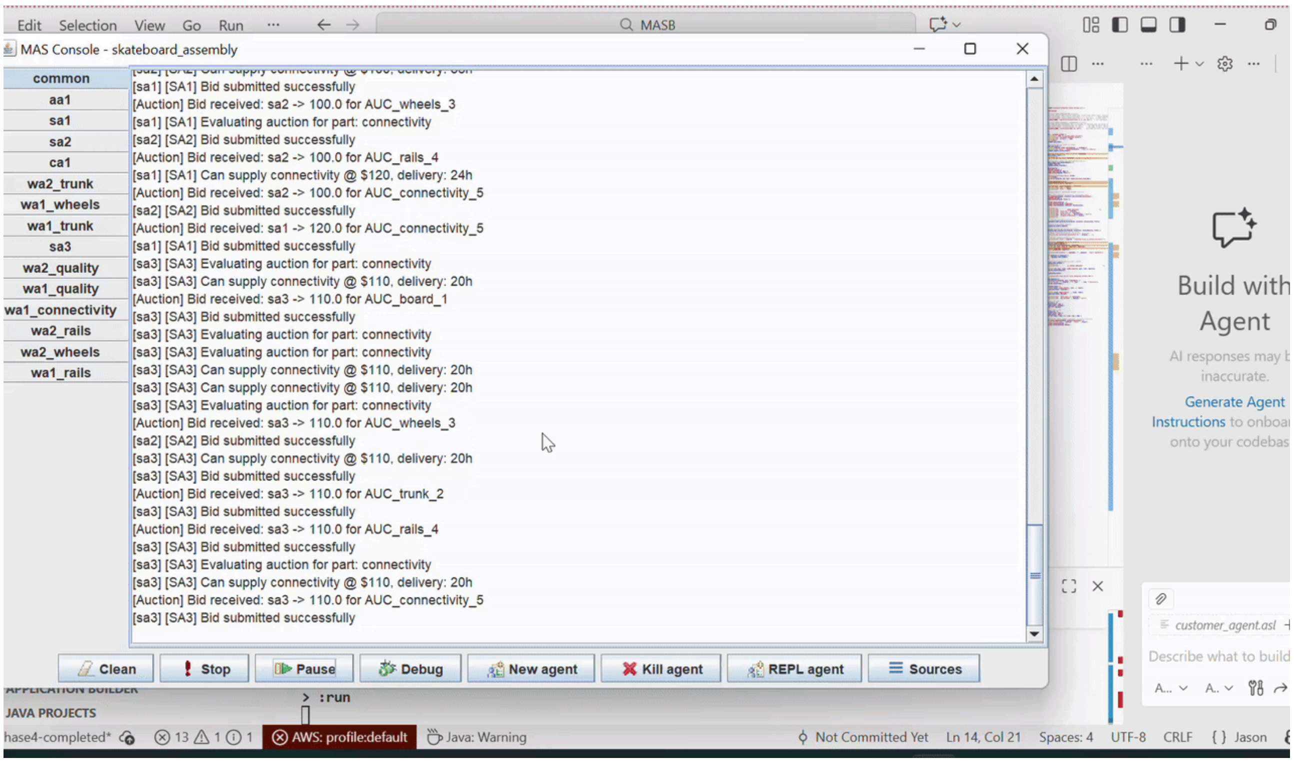
This project successfully demonstrates the application of multi-agent systems to manufacturing coordination using the JaCaMo platform. The implementation covers all four coordination paradigms: agent, environment, interaction, and organization-centered, providing a comprehensive example of MAS design principles. While certain advanced features remain unimplemented, the core system achieves its objectives of processing configurable skateboard orders through competitive procurement, resource-constrained allocation, and coordinated assembly with visual feedback. The modular architecture provides a solid foundation for future enhancements and serves as an educational example of industrial multi-agent system design.

5. Visualization

5.1 Activity Diagram

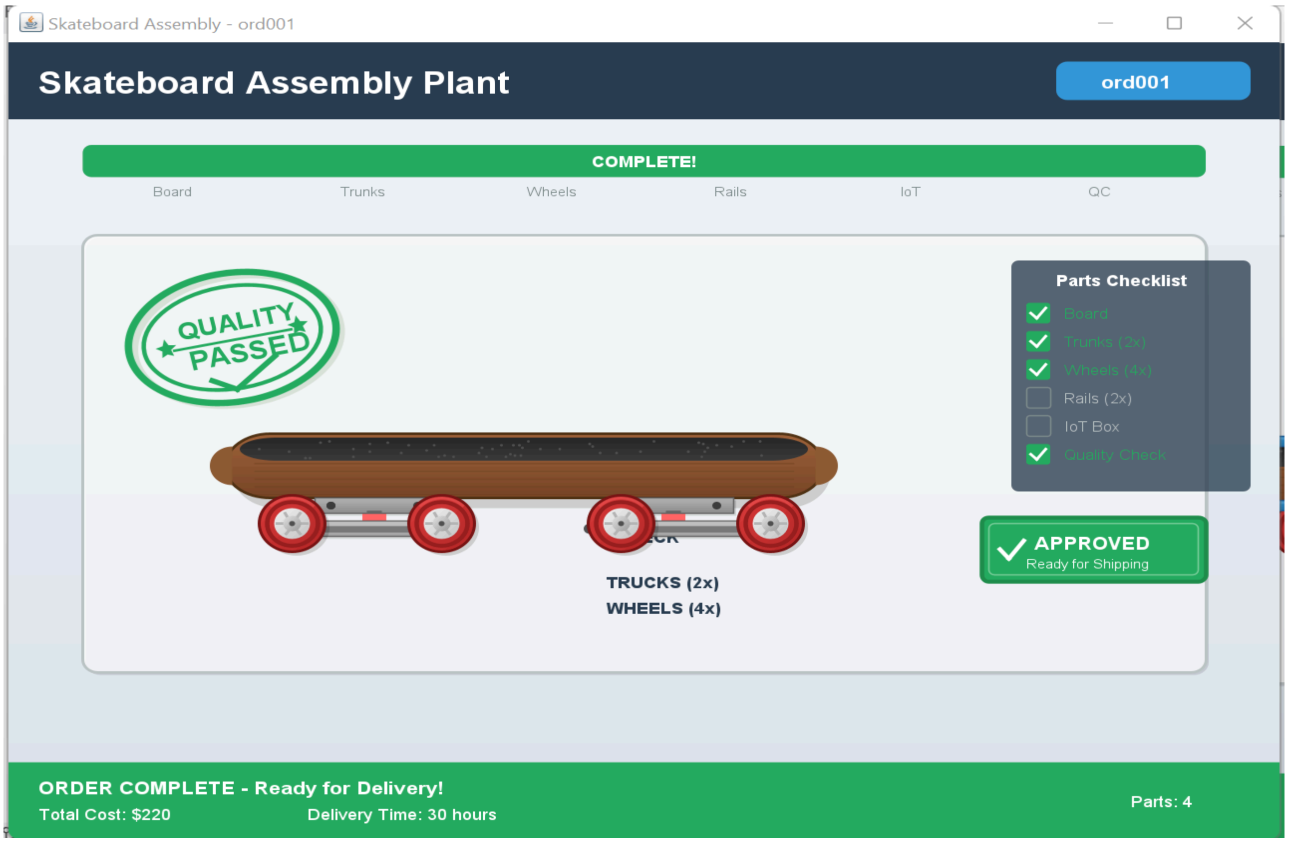


5.2 Project Visualization



```
[sa1] [SA1] Bid submitted successfully
[Auction] Bid received: sa2 -> 100.0 for AUC_wheels_3
[sa1] [SA1] Evaluating auction for part: connectivity
[sa2] [SA2] Bid submitted successfully
[Auction] Bid received: sa2 -> 100.0 for AUC_rails_4
[sa1] [SA1] Can supply connectivity @ $120, delivery: 24h
[Auction] Bid received: sa2 -> 100.0 for AUC_connectivity_5
[sa2] [SA2] Bid submitted successfully
[Auction] Bid received: sa1 -> 120.0 for AUC_connectivity_5
[sa1] [SA1] Bid submitted successfully
[sa3] [SA3] Evaluating auction for part: connectivity
[sa3] [SA3] Can supply connectivity @ $110, delivery: 20h
[Auction] Bid received: sa3 -> 110.0 for AUC_board_1
[sa3] [SA3] Bid submitted successfully
[sa3] [SA3] Evaluating auction for part: connectivity
[sa3] [SA3] Evaluating auction for part: connectivity
[sa3] [SA3] Can supply connectivity @ $110, delivery: 20h
[sa3] [SA3] Can supply connectivity @ $110, delivery: 20h
[sa3] [SA3] Evaluating auction for part: connectivity
[Auction] Bid received: sa3 -> 110.0 for AUC_wheels_3
[sa2] [SA2] Bid submitted successfully
[sa3] [SA3] Can supply connectivity @ $110, delivery: 20h
[sa3] [SA3] Bid submitted successfully
[Auction] Bid received: sa3 -> 110.0 for AUC_trunk_2
[sa3] [SA3] Bid submitted successfully
[Auction] Bid received: sa3 -> 110.0 for AUC_rails_4
[sa3] [SA3] Bid submitted successfully
[sa3] [SA3] Evaluating auction for part: connectivity
[sa3] [SA3] Can supply connectivity @ $110, delivery: 20h
[Auction] Bid received: sa3 -> 110.0 for AUC_connectivity_5
[sa3] [SA3] Bid submitted successfully
```

5.3 Minimum Skateboard



Skateboard Assembly - ord001

Skateboard Assembly Plant

ord001

COMPLETE!

Board Trunks Wheels Rails IoT QC

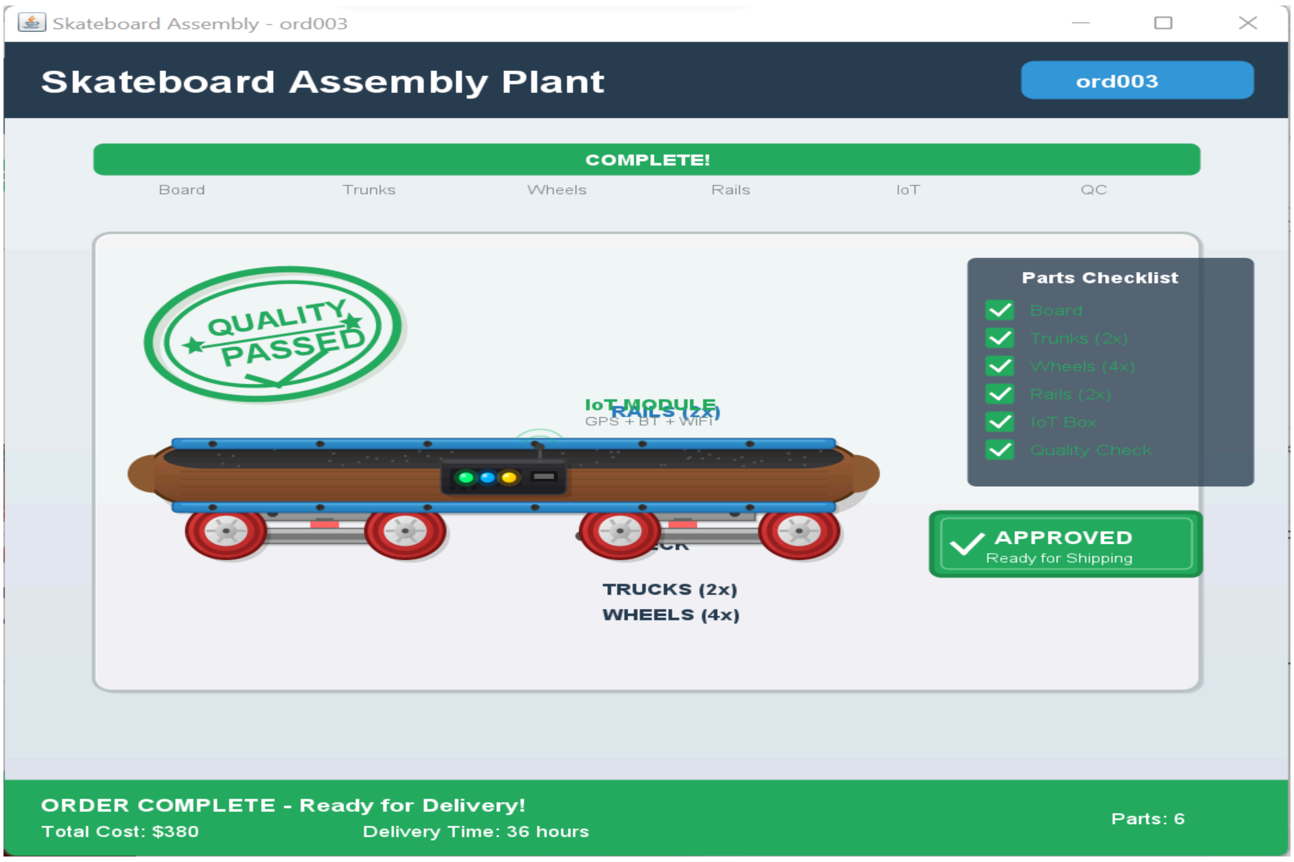
Parts Checklist

- ✓ Board
- ✓ Trunks (2x)
- ✓ Wheels (4x)
- ☐ Rails (2x)
- ☐ IoT Box
- ✓ Quality Check

APPROVED
Ready for Shipping

ORDER COMPLETE - Ready for Delivery!
Total Cost: $220 Delivery Time: 30 hours Parts: 4

5.4 Max Skateboard



Skateboard Assembly - ord003

Skateboard Assembly Plant

ord003

COMPLETE!

Board Trunks Wheels Rails IoT QC

Parts Checklist

- ✓ Board
- ✓ Trunks (2x)
- ✓ Wheels (4x)
- ✓ Rails (2x)
- ✓ IoT Box
- ✓ Quality Check

APPROVED
Ready for Shipping

ORDER COMPLETE - Ready for Delivery!
Total Cost: $380 Delivery Time: 36 hours Parts: 6