Gheorghe Asachi Technical University of Iaşi
Faculty of Automatic Control and Computer Engineering
Department of Automatic Control and Applied Informatics

DOCUMENTATION

Rule-Based Multi-Robot Planning System for Manufacturing Automation

Author
Denisa-Gabriela MUSTEATA

Iași, 2025

Contents

1 Project Description

This project implements a rule-based expert system using the CLIPS programming environment to coordinate a team of four robots in a manufacturing scenario. The robots are tasked with placing specific parts (from types A to F) in a grid of eight positions, according to a predefined manufacturing goal. Each robot has an initial location, speed, and local storage with various parts.

The system determines the optimal robot-to-goal assignments based on part availability and travel time, while also handling part shortages, avoiding robot collisions, enabling part transfers between neighboring robots, and rebalancing inventories once all goals are completed. All logic is encoded using structured facts and prioritized rules, demonstrating intelligent planning and multi-agent coordination within a constrained environment.

The expert system provides:

- Structured representation of robots, parts, goals, and spatial relationships.
- Automated assignment of robots to goals based on part availability, distance, and speed.
- Dynamic part transfers between neighboring robots to resolve shortages.
- Collision detection to avoid simultaneous actions in conflicting positions.
- Final inventory balancing after all manufacturing goals are completed.

This project not only deepens understanding of rule-based reasoning and multi-agent coordination, but also lays the groundwork for advanced production systems that integrate planning, collaboration, and decision-making within autonomous environments.

2 Knowledge Representation

2.1 Facts description

The knowledge base of the robotic manufacturing system is organized around a collection of structured facts that represent entities, actions, relationships, and the current system state. These facts are defined using templates that model robots, their positions and inventories, goal requirements, neighbor connections, assigned tasks, detected collisions, part transfers, and redistribution actions. Each fact serves as a data point in the inference process, allowing the expert system to reason about the environment, update state dynamically, and drive intelligent robot coordination through declarative rules.

In this project, all facts are defined using **`deftemplate`**, which allows for structured and semantically clear knowledge representation. By using slots and multislot constructs, the system ensures better organization of data and facilitates more precise pattern matching in rule conditions. This choice supports modularity, reduces ambiguity, and enables easier debugging compared to unstructured (ordered) facts.

`deftemplate` constructs enable the system to identify, compare, and modify information using rule-based inference. The following section outlines each fundamental fact template used in the system, providing both its structural definition and an example of how it is applied in practice.

2.1.1 Fact Templates

1. **`robot`** - this template defines each robot in the system by its unique identifier, coordinates on the workspace grid, movement speed, operational state (free or busy), and its local storage containing the available parts. It is central to task planning and execution.

   Generic template:

   ```
   (deftemplate robot
   (slot id)
   (slot x)
   (slot y)
   (slot speed)
   (slot state)
   (multislot device-storage))
   ```

   Example:

   ```
   (robot (id R1) (x 0) (y 0) (speed 1) (state free) (device-storage A B))
   ```

2. **`goal`** - this fact specifies the manufacturing objectives, identifying where a part must be placed and what type is required. Each goal is uniquely identified and corresponds to a position in the workspace.

   Generic template:

```
(deftemplate goal
(slot id)
(slot x)
(slot y)
(slot package-id))
```

Example:

```
(goal (id g1) (x 1) (y 1) (package-id A))
```

3. `assigned` - this structure links a robot to a goal it has been assigned, and includes the calculated time cost based on distance and robot speed. It reflects active planning.

   Generic template:

```
(deftemplate assigned
(slot robot-id)
(slot goal-id)
(slot time-cost))
```

   Example:

```
(assigned (robot-id R1) (goal-id g1) (time-cost 2))
```

4. `missing-part` - used to signal that a part required by one or more goals is not currently available in any robot's inventory. This warning helps prioritize transfers or user intervention.

   Generic template:

```
(deftemplate missing-part
(slot package-id))
```

   Example:

```
(missing-part (package-id C))
```

5. `collision` - captures a potential collision event between two robots that are assigned to adjacent cells in the same row. It is triggered after assignment and helps identify unsafe scheduling.

   Generic template:

```
(deftemplate collision
(slot robot1)
(slot robot2)
(slot y))
```

   Example:

```
(collision (robot1 R1) (robot2 R2) (y 1))
```

6. `part-transfer` - this fact logs a transfer of a part between two robots, typically when one robot lacks a needed item that exists in a neighbor's inventory.

   Generic template:

```
(deftemplate part-transfer
(slot from)
(slot to)
(slot part))
```

Example:

```
(part-transfer (from R2) (to R1) (part D))
```

7. **`neighbor`** - defines neighbor relationships between robots. This information is used to limit part transfers to only directly connected agents.

   Generic template:

```
(deftemplate neighbor
(slot robot-id)
(slot robot-vecin))
```

Example:

```
(neighbor (robot-id R1) (robot-vecin R2))
```

8. **`transferred`** - acts as a flag to prevent repeated part transfers for the same part and robot. It ensures stability in the inference cycle by avoiding infinite rule activations.

   Generic template:

```
(deftemplate transferred
(slot to)
(slot part))
```

Example:

```
(transferred (to R1) (part D))
```

9. **`redistribute`** - this template represents a post-goal redistribution action, meant to balance part availability among robots after all goals have been met.

   Generic template:

```
(deftemplate redistribute
(slot from)
(slot to)
(slot part))
```

   Example:

```
(redistribute (from R4) (to R3) (part B))
```

These fact templates form the structural backbone of the expert system's reasoning capabilities. They represent real-world entities and events in a symbolic form that the rule engine can match, manipulate, and reason about. By encoding the dynamic environment in this way, the system can react to changes, enforce constraints, and simulate intelligent robotic coordination in a manufacturing scenario.

2.2 Rules description

2.2.1 Optimal Robot Assignment to Goals

This task handles the assignment of robots to goals based on the most efficient plan in terms of time. The system evaluates the Manhattan distance between each robot and the goal and divides it by the robot's speed to compute a time-cost metric. Only robots that have the required package in their device storage and are free are considered. The robot is then marked as busy.

Trigger: `assign-optimal-goal` is triggered when a goal is not assigned and a compatible, available robot exists.

Rule logic outline:

```
For each unassigned goal:
Find all free robots that have the required part
Calculate time-cost = Manhattan distance / speed
Assign the robot to the goal
Mark the robot as busy
```

CLIPS Rule:

```
(defrule assign-optimal-goal
(declare (salience 50))
?g <- (goal (id ?gid) (x ?gx) (y ?gy) (package-id ?pkg))
?r <- (robot (id ?rid) (x ?rx) (y ?ry) (speed ?sp) (state free)
(device-storage \$? ?pkg \$?))
(not (assigned (goal-id ?gid)))
\=>
(bind ?dist (+ (abs (- ?gx ?rx)) (abs (- ?gy ?ry))))
(bind ?time (/ ?dist ?sp))
(printout t "ASSIGN: " ?rid " assigned to goal " ?gid " with time cost: " ?time crlf
   )
(assert (assigned (robot-id ?rid) (goal-id ?gid) (time-cost ?time)))
(modify ?r (state busy)))
```

2.2.2 Missing Part Detection

Detects when no robot possesses the necessary package for a given goal, indicating that the goal cannot be fulfilled unless redistribution occurs.

Trigger: `check-missing-part` is triggered when a goal is defined, but no robot has the package.

Rule logic outline:

```
For each goal:
If no robot has the required package:
Assert missing-part
Warn the user
```

CLIPS Rule:

```
(defrule check-missing-part
(declare (salience 100))
?g <- (goal (package-id ?pkg))
(not (robot (device-storage \$? ?pkg \$?)))
(not (missing-part (package-id ?pkg)))
\=>
(assert (missing-part (package-id ?pkg)))
(printout t "WARNING: Missing part " ?pkg " in all robot storages." crlf))
```

2.2.3 Collision Detection

Prevents planning conflicts by identifying if two robots are assigned to goals on the same Y-coordinate and their X-positions are adjacent or the same.

Trigger: `detect-collision` is triggered after multiple goals have been assigned.

Rule logic outline:

```
For each pair of assigned robots:
If their goals are on the same Y and adjacent X:
Assert collision
Print warning
```

CLIPS Rule:

```
(defrule detect-collision
(declare (salience 10))
(assigned (robot-id ?r1) (goal-id ?g1))
(assigned (robot-id ?r2) (goal-id ?g2))
(goal (id ?g1) (y ?y1) (x ?x1))
(goal (id ?g2) (y ?y2) (x ?x2))
(test (and (= ?y1 ?y2) (neq ?r1 ?r2)
(<= (abs (- ?x1 ?x2)) 1)))
\=>
(assert (collision (robot1 ?r1) (robot2 ?r2) (y ?y1)))
(printout t "WARNING: Potential collision between " ?r1 " and " ?r2 " at line Y=" ?
    y1 crlf))
```

2.2.4 Robot Part Transfer

Facilitates the redistribution of parts between neighboring robots when a robot needs a part to fulfill a goal.

Trigger: `initiate-part-transfer` is triggered when a robot lacks a required part and a neighboring robot has it.

Rule logic outline:

```
For each goal:
If a robot lacks the part but a neighbor has it:
Transfer the part
Update both 'robots device-storage
Mark the transfer to prevent duplicates
```

CLIPS Rule:

```
(defrule initiate-part-transfer
(declare (salience 90))
(goal (package-id ?pkg))
?r1 <- (robot (id ?rid1) (device-storage \$?ds1))
(test (not (member\$ ?pkg ?ds1)))
(neighbor (robot-id ?rid1) (robot-vecin ?rid2))
?r2 <- (robot (id ?rid2) (device-storage \$?before ?pkg \$?after))
(not (transferred (to ?rid1) (part ?pkg)))
\=>
(modify ?r2 (device-storage ?before \$?after))
(modify ?r1 (device-storage ?ds1 ?pkg))
(assert (transferred (to ?rid1) (part ?pkg)))
(assert (part-transfer (from ?rid2) (to ?rid1) (part ?pkg)))
(printout t "TRANSFER: Part " ?pkg " moved from " ?rid2 " to " ?rid1 crlf)
)
```

2.2.5 Redistribution After Goals Are Completed

After all goals have been fulfilled, this rule redistributes parts among robots to balance storage loads.

Trigger: `redistribute-after-goals` is triggered when no more goals remain in the system.

Rule logic outline:

```
If no more goals:
For each robot pair:
If one has significantly more parts than the other:
Transfer a part from richer to poorer
Assert a redistribution fact
```

CLIPS Rule:

```
(defrule redistribute-after-goals
  (declare (salience 0))
  (not (goal))
  ?r1 <- (robot (id ?rid1) (device-storage $?ds1))
  ?r2 <- (robot (id ?rid2) (device-storage $?ds2))
  (test (> (length$ ?ds1) (+ (length$ ?ds2) 1)))
  =>
  (bind ?part (nth$ 1 ?ds1))
  (assert (redistribute (from ?rid1) (to ?rid2) (part ?part)))
  (printout t "BALANCE: Transfer part " ?part " from " ?rid1 " to " ?rid2 crlf))
```

2.3 Testing and Debugging

To verify the correctness, stability, and logical consistency of the robot coordination expert system, we applied a structured testing strategy covering individual rules and integrated scenarios:

- **Unit-Level Testing of Rules**
  Each rule (e.g., `assign-optimal-goal`, `initiate-part-transfer`, `detect-collision`) was tested independently by asserting only the necessary facts. For example, to test the part transfer, a robot missing a required part was paired with a neighbor having the part, and the rule's behavior was observed.

- **Scenario-Based Testing**
  Full simulations were conducted using the predefined `test-database` with multiple robots and goals. Scenarios included goal assignment, missing part resolution, inter-robot transfers, collision alerts, and final redistribution. This allowed evaluation of multi-rule interaction over time.

- **Edge Case Handling**
  The system was evaluated with special cases such as:

  - no robot has the required part for a goal;
  - two robots are assigned to conflicting positions;
  - repeated transfers without control flags;
  - uneven part distribution post-mission.

  These tested the system's defensive logic and prevented infinite rule firings or invalid actions.

- **State Verification**
  After each rule execution, the working memory (facts) was printed using '(facts)' and rule activity was monitored via '(watch rules)' and '(watch facts)'. This confirmed that each robot updated its storage correctly, transfers occurred only once, and goals were assigned optimally.

Through this testing approach, the system was proven to operate reliably under varied conditions, validating rule priority, agent behavior, and coordination logic in a simulated manufacturing environment.

3 Results

This expert system for autonomous robot coordination showcases how rule-based reasoning can be effectively applied to manage tasks in a dynamic, distributed manufacturing environment. By leveraging declarative knowledge representation and forward-chaining inference, the system automates goal assignment, detects missing resources, enables inter-robot part transfers, prevents collisions, and redistributes parts after task completion.

The system is designed with modularity and transparency in mind. Each functionality—whether it is optimal robot-goal assignment, real-time part balancing, or conflict detection—is implemented through clear rule constructs and structured facts. This ensures interpretability of robot decisions and provides flexibility for future scalability or domain adaptation.

From an AI perspective, the project demonstrates the strength of symbolic reasoning and expert systems in multi-agent coordination and logistics. It reinforces key concepts such as state-driven rule triggering, fact matching, and dynamic updates to a shared working memory. These mechanisms collectively simulate intelligent cooperation between autonomous agents with limited resources.

The project provides a functional and extensible prototype that models realistic factory-floor decision logic. Potential future enhancements could include dynamic goal injection, priority-based planning, spatial obstacle handling, energy/resource monitoring, or real-time integration with physical robotic systems. As implemented, the expert system provides a solid foundation for intelligent task planning and resource coordination in industrial settings.

During testing, the system proved capable of adapting to scenarios where required parts were initially unavailable, successfully initiating redistribution and transfer between neighboring robots. The detection of critical conditions, such as potential collisions or unresolvable goals due to missing parts, enhances both safety and predictability in a coordinated multi-agent environment.

Additionally, the redistribution mechanism triggered after task completion ensures that resources are balanced among robots for future operations. This post-goal optimization step adds a layer of intelligence that simulates adaptive logistics behavior, maintaining readiness across the entire robot network for incoming assignments.

4 Optional Enhancements

While the current implementation meets the core functional requirements of task allocation and resource management, the system could be extended in several ways to improve realism and adaptability.

- **Priority-Based Goals**
  Allowing certain goals to have higher priority would enable more strategic planning, ensuring that critical deliveries are handled first.
- **Energy Constraints and Recharge Logic**
  Introducing energy levels for each robot would simulate real-world limitations and require planning for recharging or load balancing.
- **Dynamic Goal Addition**
  Supporting real-time injection of new goals during execution would reflect changing demands in an industrial setting and test the system's adaptability.
- **Graphical Output or Simulation**
  A simple visualization layer could help track robot movement, transfers, and goal completion, making the system more demonstrable and easier to debug.

5 Conclusion

This expert system for autonomous robot coordination demonstrates how symbolic AI and rule-based reasoning can be effectively applied in managing task assignment, resource distribution, and conflict prevention in a multi-agent environment. Using a declarative knowledge base and forward-chaining inference, the system assigns robots to tasks based on resource availability and location efficiency, while also handling part transfers, collision avoidance, and storage balancing.

The system's architecture is modular and interpretable, relying on clearly defined templates and rules that reflect practical behaviors found in industrial robotics and logistics. By working with structured facts and explicitly encoded rules, the system ensures traceability, ease of debugging, and extendability to more complex manufacturing environments.

Beyond functional automation, the project illustrates core principles in knowledge representation, inference control, and fact-based decision making. It showcases how expert systems can simulate intelligent behavior among multiple agents that collaborate under constraints, making it a useful model for educational, research, or prototyping purposes.

Future development could incorporate real-time updates, energy consumption tracking, dynamic prioritization of tasks, or integration with robotic simulators or hardware. In its current form, the system provides a clear and functional prototype for intelligent coordination and task optimization across distributed robotic agents.

Bibliography

[1] Joseph Giarratano, Gary Riley. *Expert Systems: Principles and Programming.* Cengage Learning, 4th Edition, 2004. ISBN: 978-0534384474.

[2] Ron Brachman, Hector Levesque. *Knowledge Representation and Reasoning.* Elsevier, Amsterdam, 2004.

[3] Stuart Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach.* Fourth Edition, Pearson, 2021.

[4] Claudia I. Rusu. *Introduction to CLIPS – Lecture Notes.* Faculty of Computer Science, Alexandru Ioan Cuza University, 2018. `https://profs.info.uaic.ro/čires/clips.html`

[5] E. Turban, J. Aronson. *Decision Support and Expert Systems.* Prentice Hall, 1998.