Multi-Agent Goal Reasoning with the CLIPS Executive in the RoboCup Logistics League

ICAART'21

Till Hofmann, Tarik Viehmann, Mostafa Gomaa,
Daniel Habering, Tim Niemueller, Gerhard Lakemeyer

Carologistics RoboCup Team

February 4, 2021

Planning and Execution on Mobile Robots

Challenges when reasoning on mobile robots:

- Offline planning is not sufficient
- Robots need to react to a changing environment and adapt their goals
- Action execution is unreliable:
  - Actions may fail
  - Plans may execute faster/slower than expected
- Robots need to coordinate effectively

Planning and Execution on Mobile Robots

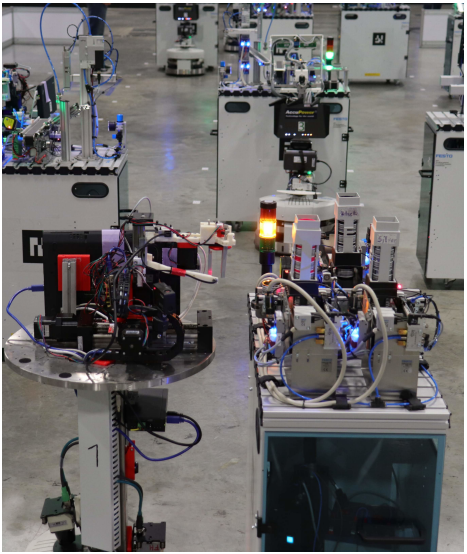
Challenges when reasoning on mobile robots:

- Offline planning is not sufficient
- Robots need to react to a changing environment and adapt their goals
- Action execution is unreliable:
  - Actions may fail
  - Plans may execute faster/slower than expected
- Robots need to coordinate effectively

Multi-Agent Goal Reasoning with the CLIPS Executive in the RoboCup Logistics League

The RoboCup Logistics League (RCLL)

- Replication of a smart factory
- In-factory production logistics
- Robots operate machines to manufacture products
- Machines are static, robots are mobile
- Each machine performs one manufacturing step
- Each team consists of three robots



RoboCup Logistics League – Production

Product Composition:

- Products of four complexities (number of rings)
- Base (3 colors) + 0–3 rings (4 colors) + cap (2 colors)
- Machines:

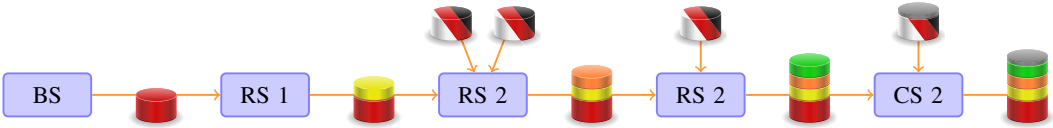
  Base Station: retrieve base workpiece
  Ring Station: mount ring
  Cap Station: mount cap
  Delivery Station: deliver product

RoboCup Logistics League – Production

Product Composition:

- Products of four complexities (number of rings)
- Base (3 colors) + 0–3 rings (4 colors) + cap (2 colors)
- Machines:

  Base Station: retrieve base workpiece
  Ring Station: mount ring
  Cap Station: mount cap
  Delivery Station: deliver product



Goal Reasoning with the CLIPS Executive [Niemueller et al., 2019]

Goal reasoning:

- No fixed goal
- Explicitly represent goals
- Continually reason about goals
- Dynamically adjust and prioritize goals

→ Reason about *what* to accomplish,
only then *how* to accomplish it

- Goal lifecycle: goal progress during execution

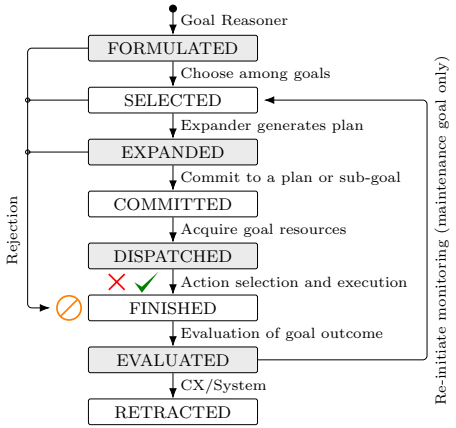
Goal Reasoning with the CLIPS Executive [Niemueller et al., 2019]

Goal reasoning:

- No fixed goal
- Explicitly represent goals
- Continually reason about goals
- Dynamically adjust and prioritize goals

→ Reason about *what* to accomplish, only then *how* to accomplish it

- Goal lifecycle: goal progress during execution



Goal Reasoning in the RCLL

Main questions:

1. How to represent the RCLL as a goal reasoning problem?
2. How to achieve one specific goal?
3. How to coordinate the three robots?

Splitting the RCLL productions into goals

- Observation: tasks can be split up to start and stop at a machine

Splitting the RCLL productions into goals

- Observation: tasks can be split up to start and stop at a machine
- A *transport* goal moves a workpiece from A to B

Splitting the RCLL productions into goals

- Observation: tasks can be split up to start and stop at a machine
- A *transport* goal moves a workpiece from A to B

Purpose	Goal Class	Source	Destination
Assembly	`MOUNT-RING`	$BS_{I/O}$, RS_O	RS_I
Assembly	`MOUNT-CAP`	RS_O	CS_I
Assembly	`PRODUCE-C0`	$BS_{I/O}$	CS_I
Preparation	`FILL-CAP`	CS_{Shelf}	CS_I
Preparation	`FILL-RS`	$BS_{I/O}$, CS_O	RS_{Slide}
Preparation	`CLEAR-MPS`	Any MPS_O	-
Completion	`DELIVER`	CS_O	DS_I

Splitting the RCLL productions into goals

- Observation: tasks can be split up to start and stop at a machine
- A *transport* goal moves a workpiece from A to B

Purpose	Goal Class	Source	Destination
Assembly	`MOUNT-RING`	$BS_{I/O}$, RS_O	RS_I
Assembly	`MOUNT-CAP`	RS_O	CS_I
Assembly	`PRODUCE-C0`	$BS_{I/O}$	CS_I
Preparation	`FILL-CAP`	CS_{Shelf}	CS_I
Preparation	`FILL-RS`	$BS_{I/O}$, CS_O	RS_{Slide}
Preparation	`CLEAR-MPS`	Any MPS_O	-
Completion	`DELIVER`	CS_O	DS_I

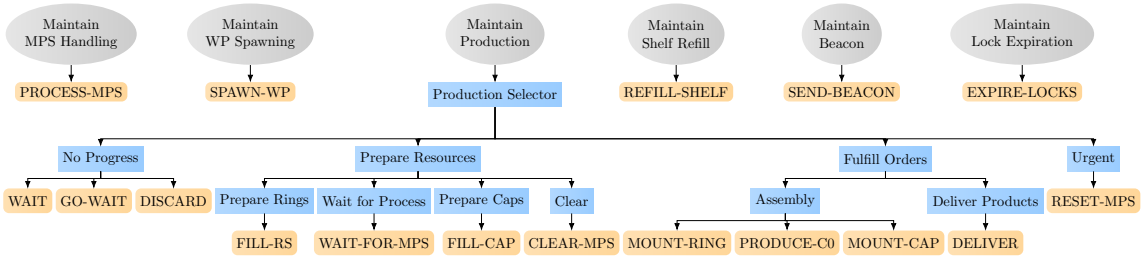
- *Machine instructions* can be handled separately
  $\rightarrow$ separate `PREPARE` goal that can be executed concurrently

Goal Trees

- *Simple* goals can be structured in a goal tree
- Goal types:
  - `RUN-ALL` Run all sub-goals successively, succeed if all sub-goals succeed
  - `TRY-ALL` Subsequently run all sub-goals until first succeeds
  - `RUN-ONE` Run one of the sub-goals, outcome is the child's outcome

Goal Trees

- *Simple* goals can be structured in a goal tree
- Goal types:
  - `RUN-ALL` Run all sub-goals successively, succeed if all sub-goals succeed
  - `TRY-ALL` Subsequently run all sub-goals until first succeeds
  - `RUN-ONE` Run one of the sub-goals, outcome is the child's outcome



Goal Expansion and Execution using PDDL

- Each goal is expanded into a PDDL plan
- PDDL preconditions are used to verify the expected world state
- PDDL effects update the agent's world model
- Execution monitoring may intervene if
  - an action is not executable
  - an action did not have the expected effects
  - the current plan or goal is no longer feasible

Goal Expansion and Execution using PDDL

- Each goal is expanded into a PDDL plan
- PDDL preconditions are used to verify the expected world state
- PDDL effects update the agent's world model
- Execution monitoring may intervene if
  - an action is not executable
  - an action did not have the expected effects
  - the current plan or goal is no longer feasible

```
(move (wait-pos ?ds INPUT) ?ds INPUT)
(wp-put ?wp ?ds)
(prepare-ds ?ds ?order)
(fulfill-order ?complexity)
(go-wait ?ds INPUT (wait-pos ?ds INPUT))
```

The plan for a `DELIVER` goal

Multi-Agent Coordination

- Distributed agent: one agent per robot

→ Each robot acts on its own

→ Need to coordinate

- Three coordination mechanisms:
  1. Shared world model
  2. Locking actions
  3. Goal resource allocation

Shared world model

- Objective: share (some) world model facts among the agents
- Fact-based world model with key-value pairs
- PDDL domain facts are translated into a key-value pair:

```
; PDDL representation
(wp-at WP1 C-CS1 INPUT)
; CLIPS representation
(domain-fact (name wp-at) (param-values WP1 C-CS1 INPUT))
; Translation into key-value pair
(wm-fact (id "/wp-at?wp=WP1&m=C-CS1&side=INPUT")
         (value TRUE))
```

- Each agent runs a MongoDB database instance
- Key-value pairs are stored in the database
- Database instances are managed as replica set
- Replica set is automatically synchronized in the background

Mutual Exclusion Mechanism

Mutual exclusion using a database:

- Separate database for *mutex documents*
- Updates using *majority acknowledgement*

→ No two agents can acquire a mutex simultaneously

Mutex expiration to avoid a crashed robot holding a mutex:

- A mutex expires after 30 s

→ After a crash, another robot can take over after the mutex expired

- Periodic refresh to avoid expiration

Locking Actions

```
(:action lock
 :parameters (?name - object)
 :precondition
  (not (locked ?name))
 :effect (locked ?name))
```

```
(:action unlock
 :parameters (?name - object)
 :precondition
  (locked ?name)
 :effect (not (locked ?name)))
```

- Locking actions based on mutexes
- Principle: lock any physical object for any interaction
- Agent waits for a certain time if it cannot acquire the lock immediately
- Specialized action `location-lock`
  - unlocking is only done after reaching a certain distance to the location

Locking Actions

```
(:action lock
 :parameters (?name - object)
 :precondition
  (not (locked ?name))
 :effect (locked ?name))
```

```
(:action unlock
 :parameters (?name - object)
 :precondition
  (locked ?name)
 :effect (not (locked ?name)))
```

- Locking actions based on mutexes
- Principle: lock any physical object for any interaction
- Agent waits for a certain time if it cannot acquire the lock immediately
- Specialized action `location-lock`
  - unlocking is only done after reaching a certain distance to the location

```
(move (wait-pos ?ds INPUT) ?ds INPUT)

(wp-put ?wp ?ds)
(prepare-ds ?ds ?order)
(fulfill-order ?complexity)


(go-wait ?ds INPUT (wait-pos ?ds INPUT))
```

Locking Actions

```
(:action lock
 :parameters (?name - object)
 :precondition
  (not (locked ?name))
 :effect (locked ?name))
```

```
(:action unlock
 :parameters (?name - object)
 :precondition
  (locked ?name)
 :effect (not (locked ?name)))
```

- Locking actions based on mutexes
- Principle: lock any physical object for any interaction
- Agent waits for a certain time if it cannot acquire the lock immediately
- Specialized action `location-lock`
  - unlocking is only done after reaching a certain distance to the location

```
(go-wait ?start ?side (wait-pos ?ds INPUT))
(location-lock ?ds INPUT)
(move (wait-pos ?ds INPUT) ?ds INPUT)
(lock ?ds)
(wp-put ?wp ?ds)
(prepare-ds ?ds ?order)
(fulfill-order ?complexity)
(unlock ?ds)
(location-unlock ?ds INPUT)
(go-wait ?ds INPUT (wait-pos ?ds INPUT))
```

Goal Resource Allocation

- Locking actions are scoped locally, i.e., to guard parts of a plan
- Expectation: The state of the resource is not altered

→ Can perform the same actions on the resource

- Not suitable for deciding *what to do*,
  e.g., buffering a cap changes the state of the CS

Goal Resource Allocation

- Locking actions are scoped locally, i.e., to guard parts of a plan
- Expectation: The state of the resource is not altered

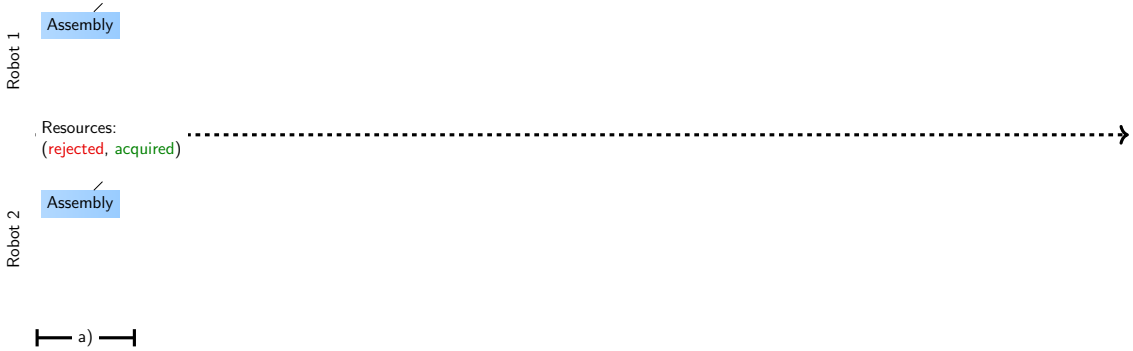
→ Can perform the same actions on the resource

- Not suitable for deciding *what to do*,
  e.g., buffering a cap changes the state of the CS

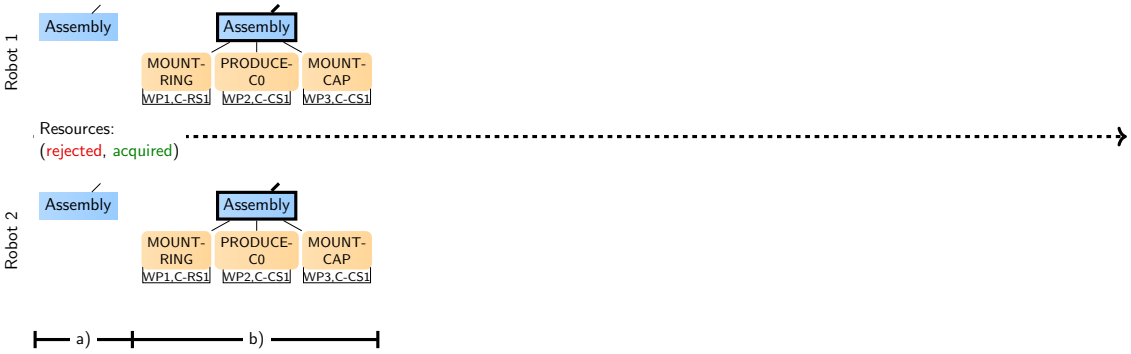
⇒ *Goal resource allocation*:
Additional coordination mechanism on goal level

- Each goal has a set of *required resources*
- Agent needs to acquire those resources before dispatching the goal

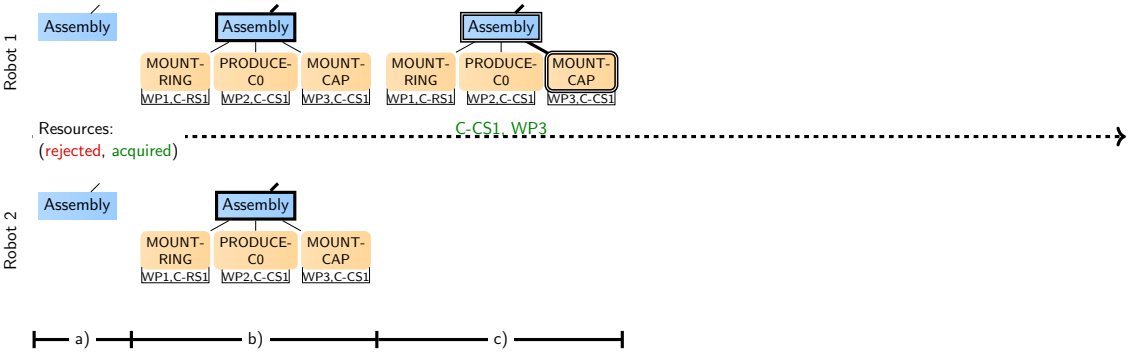
Coordination by Means of Goal Resource Allocation



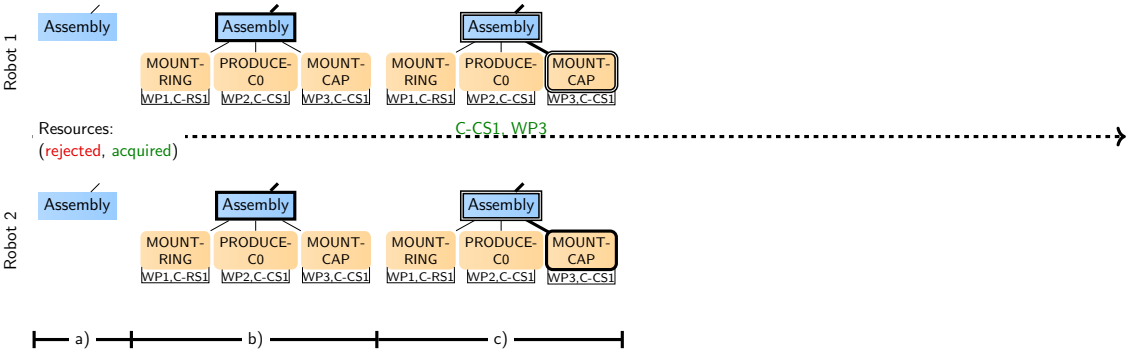
Coordination by Means of Goal Resource Allocation



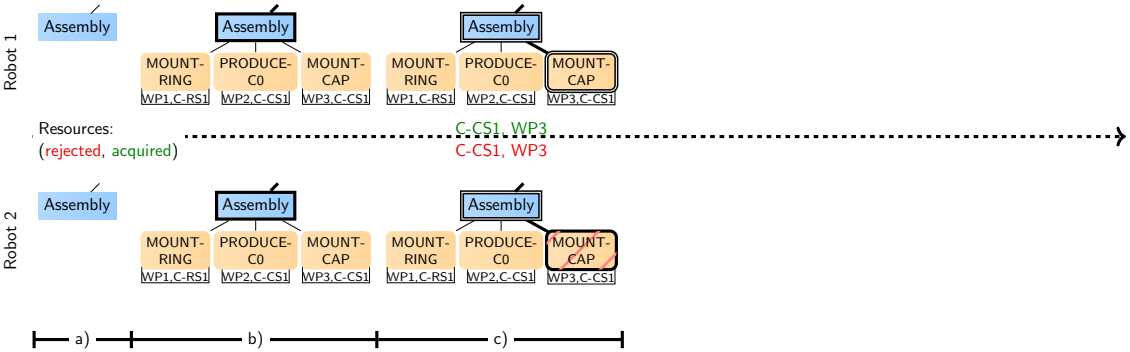
Coordination by Means of Goal Resource Allocation



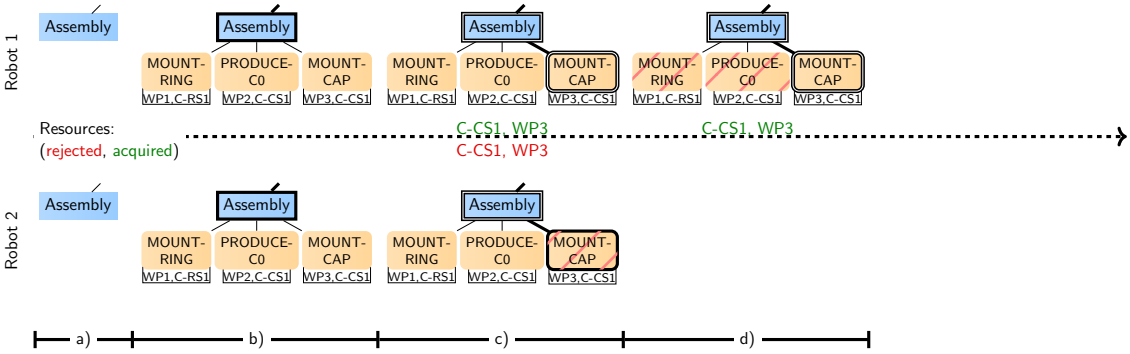
Coordination by Means of Goal Resource Allocation



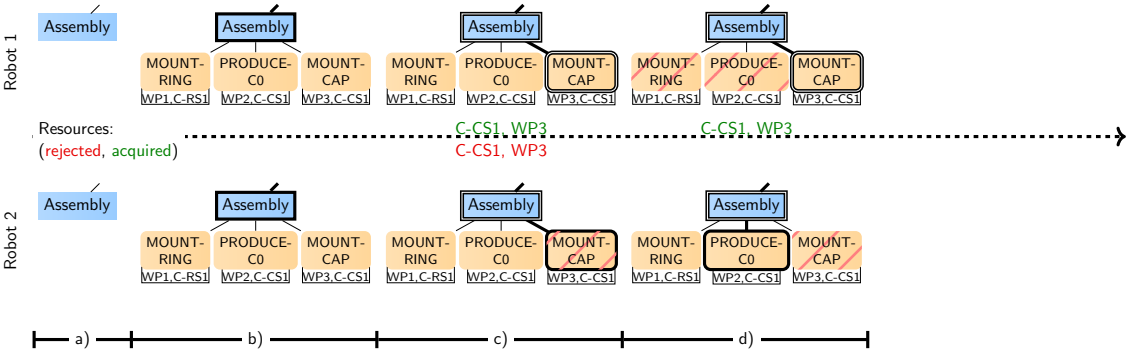
Coordination by Means of Goal Resource Allocation



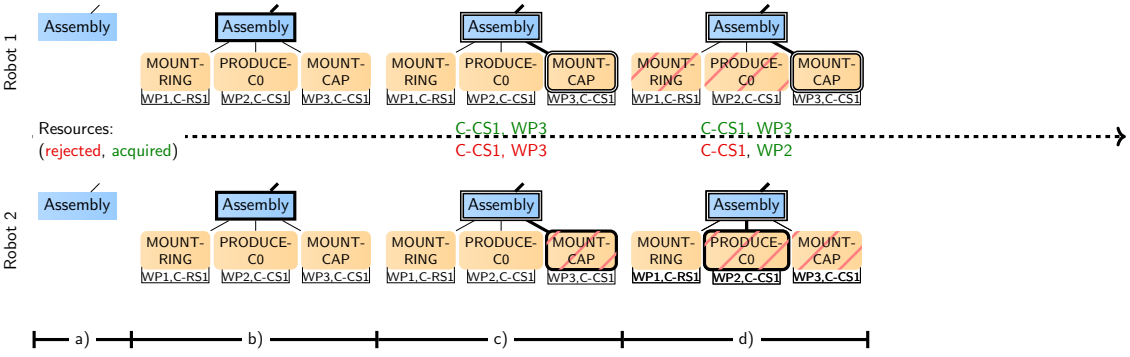
Coordination by Means of Goal Resource Allocation



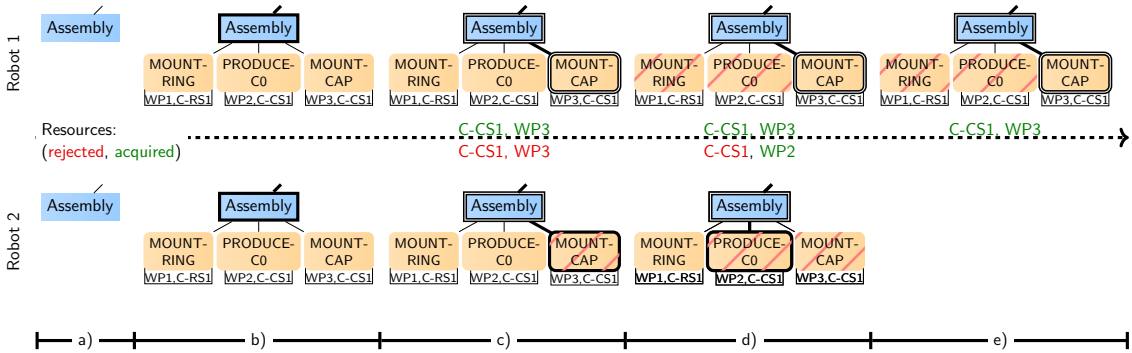
Coordination by Means of Goal Resource Allocation



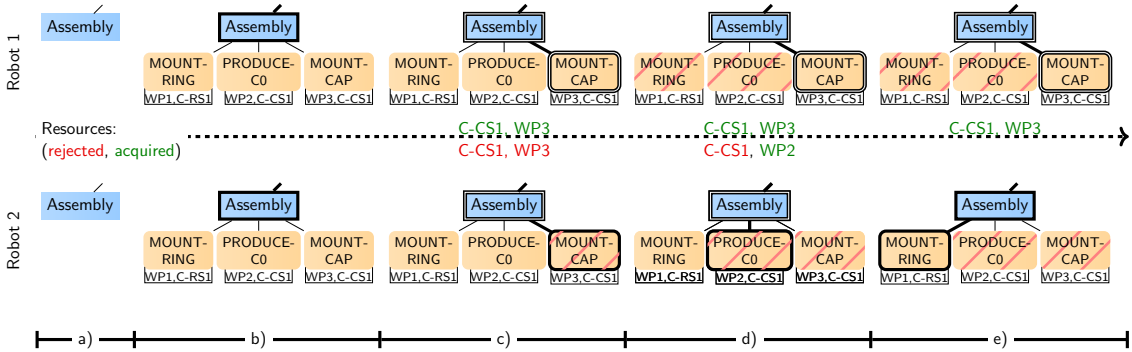
Coordination by Means of Goal Resource Allocation



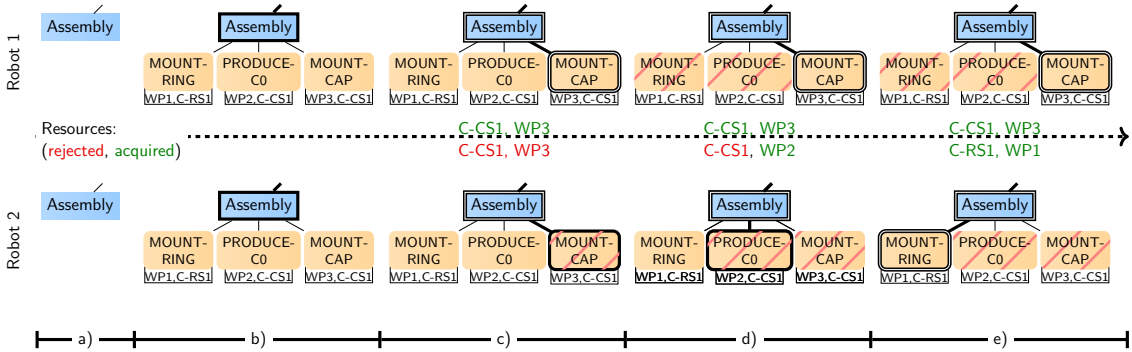
Coordination by Means of Goal Resource Allocation



Coordination by Means of Goal Resource Allocation

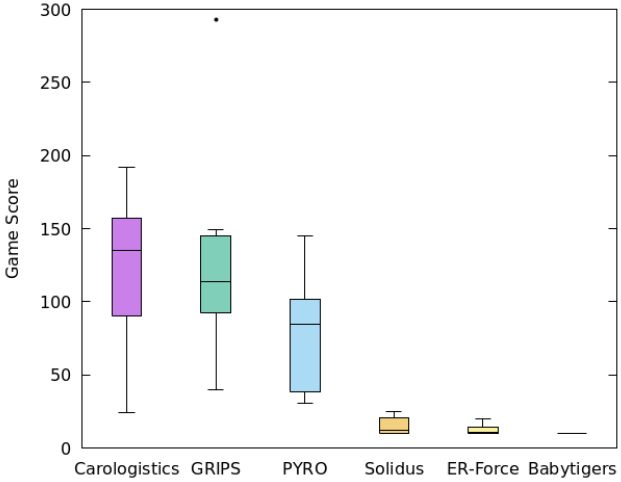


Coordination by Means of Goal Resource Allocation



Evaluation: Game Scores at RoboCup 2019

- Setting: RoboCup World Cup 2019
- Scores from all official games
- Final result: 1st place



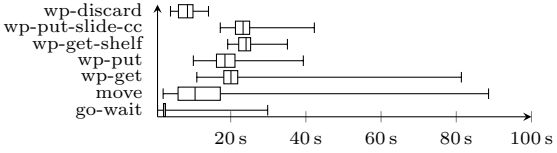
Evaluation: Execution

- High variation in execution time

→ Distributed and incremental reasoning can react dynamically

- 28 % of the goals eventually failed
- 22 % of the failed goals could be recovered
- Robots can react by retrying and taking over

⇒ Good performance despite high failure rates



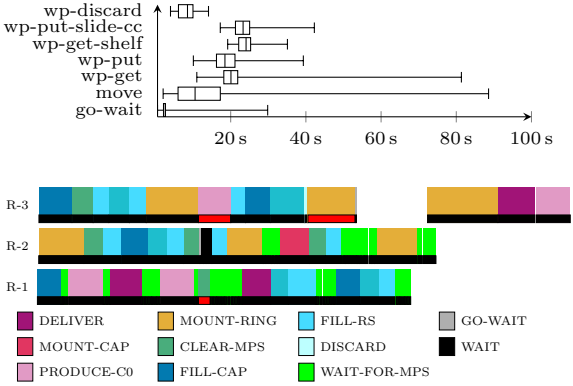
Evaluation: Execution

- High variation in execution time

→ Distributed and incremental reasoning can react dynamically

- 28 % of the goals eventually failed
- 22 % of the failed goals could be recovered
- Robots can react by retrying and taking over

⇒ Good performance despite high failure rates



Conclusion

- Distributed multi-agent goal reasoning system
- Each agent decides on its own goals
- Decide what to do next without creating a plan for the complete problem

→ Quickly react to changes during execution

- Multi-agent coordination based on:
  1. A shared world model
  2. Locking actions
  3. Goal resource allocation

→ Effective coordination of a team of three robots

References I

T. Niemueller, T. Hofmann, G. Lakemeyer.

Goal reasoning in the CLIPS Executive for integrated planning and execution.

Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS), 2019.