



# GRAN SASSO SCIENCE INSTITUTE

## Democratizing the programming and use of Robots

Patrizio Pelliccione

Director of the Computer Science area

Full professor at GSSI

Adjunct professor at the University of Bergen, Norway

[patrizio.pelliccione@gssi.it](mailto:patrizio.pelliccione@gssi.it)  
<https://www.patriziopelliccione.com/>

[www.gssi.it](http://www.gssi.it) A row of social media icons: Facebook, Twitter, LinkedIn, Instagram, YouTube, and a white square icon.

# Why Democratizing?

- **Accessibility:** technology (and robotics) extends to an ever-broader audience, and in some cases to the entire society
- **User-friendliness:** easier to use so that more people can use them (correctly and confidently) without needing advanced skills or training

# Robots in hotels

- **Hotel concierge:** specifying what the robot should do
- **Hotel guests:** interact with the robots to give feedback or to make additional requests
- **Humans in the hotel:** share the environment with robots

Flyzoo Hotel - Alibaba Future Hotel Hangzhou

<https://flyzoo-hotel.hangzhouhotel.org/en/>



# Autonomous car

- **Ride specification:** more complex than just specify the destination address
- **Degree of automation:** partial automation can be more stressful than fully manual driving, as drivers need to constantly monitor whether the vehicle is doing what it is supposed to



# Industrial Robots

- **Experts in satellite production:** specifying what the robot should do
- **Customization in the production islands:** self-contained, flexible manufacturing unit with its own specificity that operates independently while still integrating with the larger smart factory ecosystem

## PRESS RELEASES

### Thales Alenia Space unveils project to develop Space Smart Factory, one of the largest facilities of its kind in Europe

Available in [IT](#) [FR](#) [ES](#)

[IMG](#) [PDF](#)

NOV 9 2023

f t in

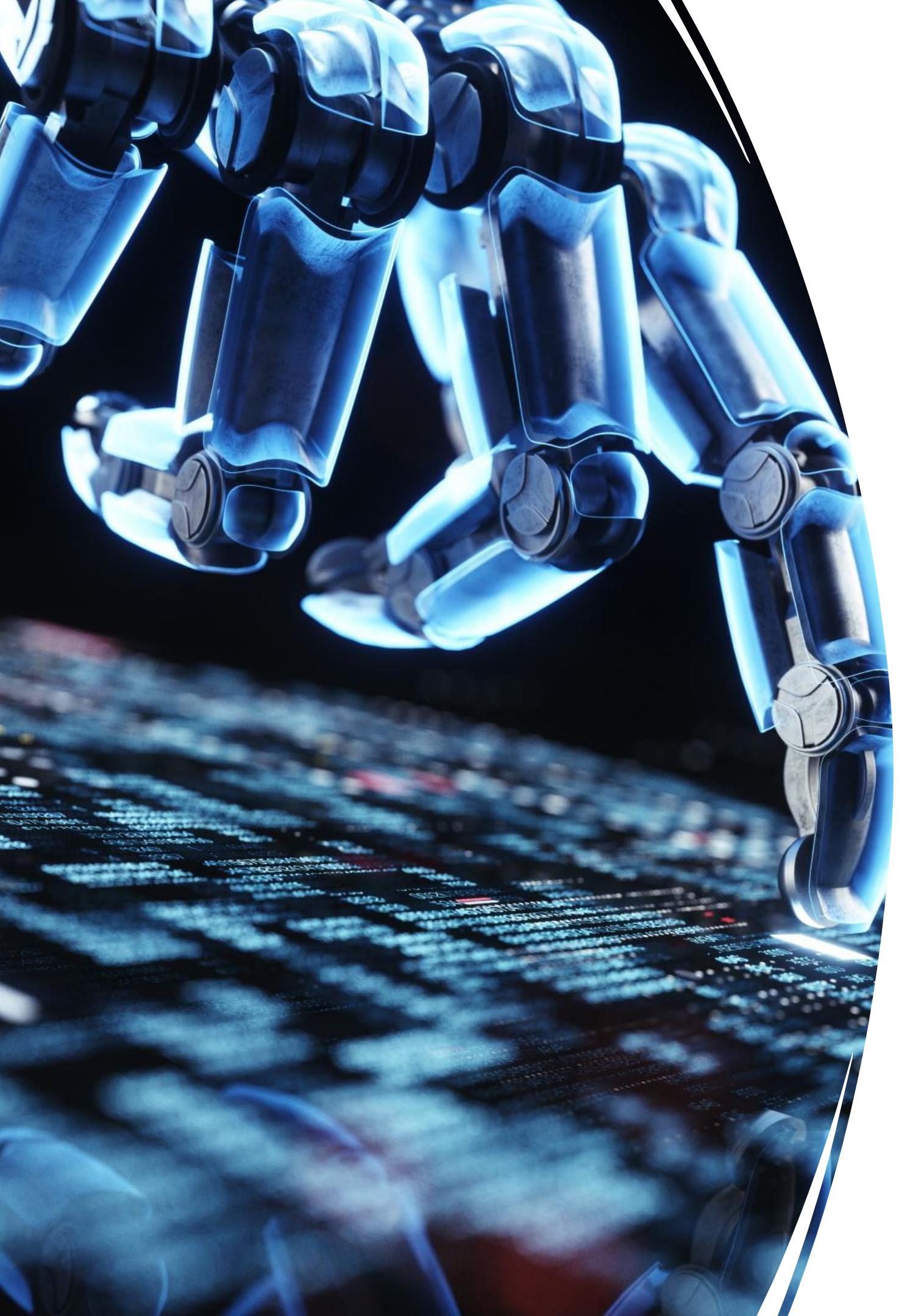


*At the Tecnopolis Tiburtino hub in Rome, Thales Alenia Space's all-digital factory will employ advanced technologies for the production of satellites*

- *The factory will be built thanks to an important investment by Thales Alenia Space and co-funded by the Italian Space Agency (ASI) through the National Recovery and Resilience Plan (PNRR) funds*
- *It will make intensive use of digital and Industry 4.0 technologies*
- *The factory will feature the Space JOINTLAB, an innovative and collaborative space with SMEs and research centers*
- *Total surface area 21,000 sq.m, 5,000 sq.m of reconfigurable clean rooms, 1,900 sq.m of office space and co-working areas, 1,800 sq.m of technical support areas*

# Why democratization and not just user friendliness?

Different degree and complexity of interaction  
that get close to programming

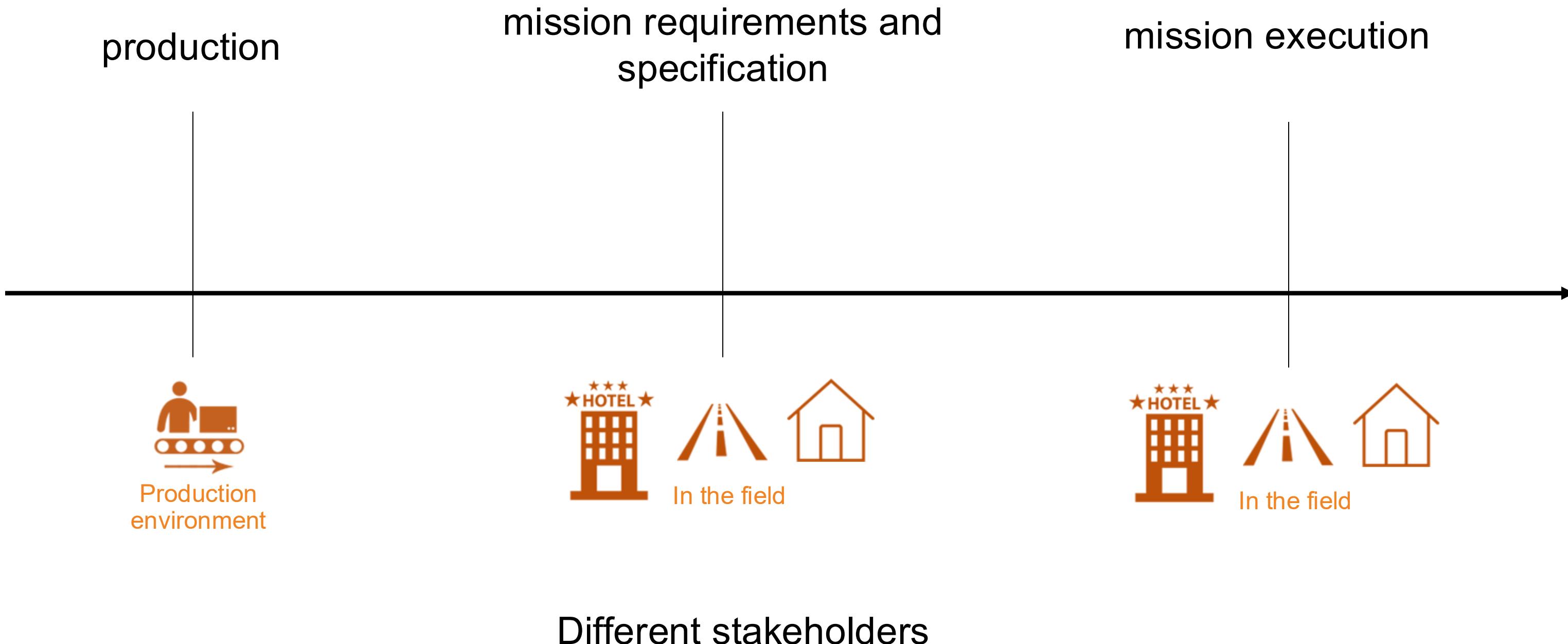


# Robotic Mission

---

- A **mission requirement** describes the high-level tasks that a robotic software must accomplish.
- A **mission specification** is a formal and precise description of what robots should do in terms of movements and actions.
- **Robotic mission engineering** concerns expressing robotic missions in high-level and user-friendly notation (mission requirements), and then translating mission requirements into more precise mission specifications.

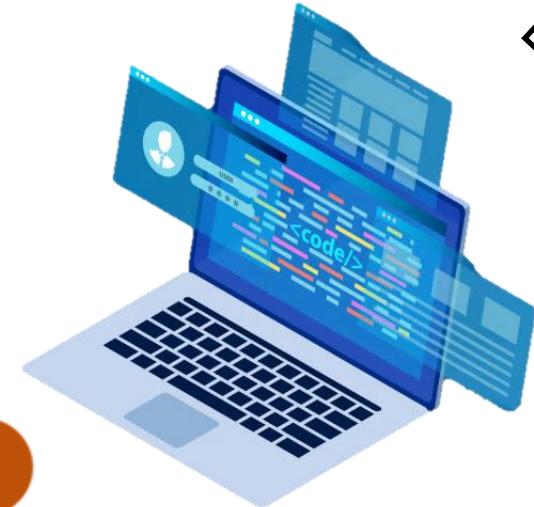
# Robots programming: beyond production environment



# Mission Specification



Production  
environment



Team of developers produce  
SAs that might include  
hardware, software, and  
mechanics

deploy



—————



How to ask the  
robot to make a  
coffee and clean  
the kitchen?

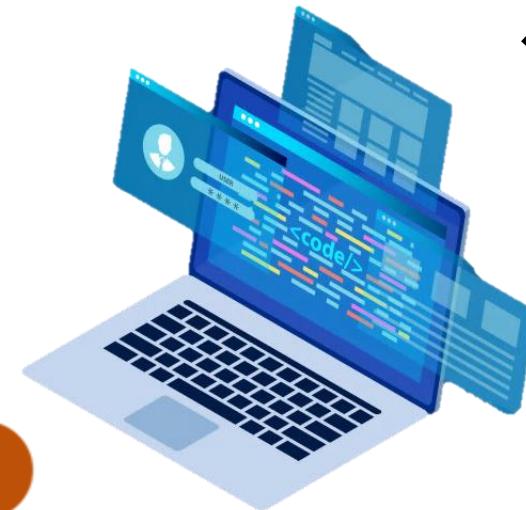


In the field

# Mission Specification



Production  
environment



deploy

Team of developers produce  
SAs that might include  
hardware, software, and  
mechanics



Programming extends in the field



How to ask the  
robot to make a  
coffee and clean  
the kitchen?



In the field

# Need of Turn-key solutions



The definition of the mission should be done in an easy and user-friendly way, accessible by users without expertise in ICT or robotic

Different stakeholders, experts of the domain but not in robotics

# Example of mission requirement

Different modes, plus dealing with the variability of the real world

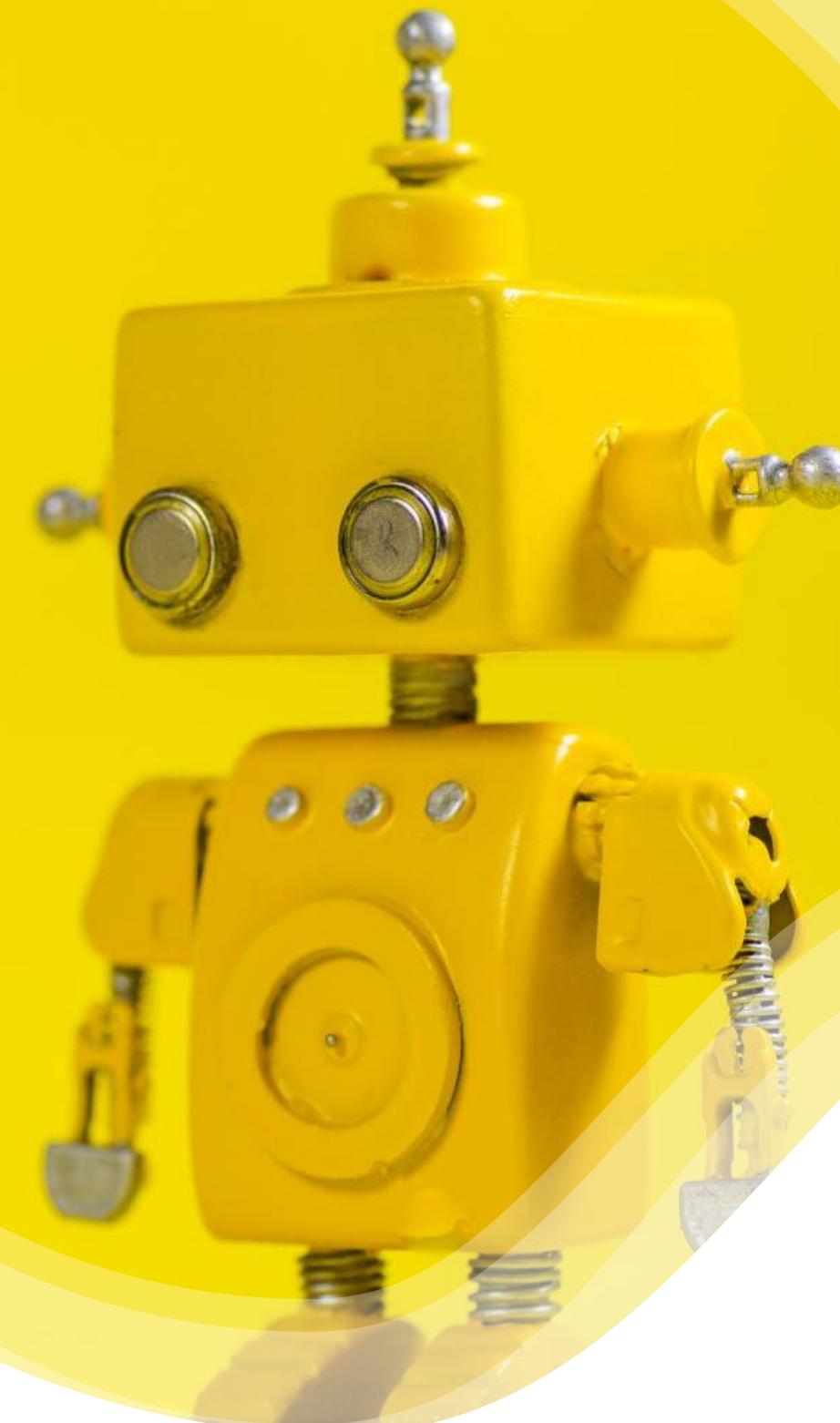
The robot should **move around** the room and **dispense medication** to independent people. It first **establishes a short conversation** based on the user's conditions to figure out the overall health status, and afterwards it will **dispense pills** along with a glass of water. The robot also **records the activity** to allow a caregiver to evaluate if the persons accepted the pills, by means of a subsequent interaction. **During night-time** a service robot performs a **cleaning protocol with the UV-lamp** on the exposed surfaces (e.g. table and chairs), possibly in **coordination with automatic cleaners** that wash the floor.

In addition, the robot should **perform regular check-ups** on people with particular conditions during free-time. It **needs to understand basic requests** and will alert the nurse in case of need, pose basic riddles or show simple pictures to test baseline human capabilities, ask the persons about their status and if they need help or assistance. Through specific questionnaires the **robot gives advices about common pathologies** affecting elderly people such as heart failure or diabetes. Tasks in hospitals are very specialized and follow very strict protocols. For these reasons, as well as efficiency, patients might stay alone for long periods of time, causing them distress and confusion. Children are a particularly affected group, as their attachment to parents is high and it is difficult for them to understand the situation.

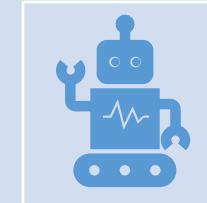
+ recharge battery when needed, deal with obstacles, presence of humans, failures, etc.

Exemplars: <https://github.com/Askarpour/RoboMAX>  
Video: <https://www.youtube.com/watch?v=txZCcABkycQ>

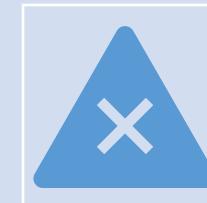
# Variability and uncertainty



What we learn from  
practitioners



It's not difficult to specify what the robot should do, i.e., the "normal" behavior.



The difficult part is to deal with uncertainty and exceptional behaviors while guaranteeing safety and the mission satisfaction.

# Which type of variability?

<b>Environment</b> Obs 1: Environment events Obs 2: Environment features Obs 3: Inclusion of humans	<b>RQ1: Drivers of variability</b>  <b>Robot Hardware</b> Obs 4: Services and capabilities Obs 5: Hardware customization impact  Obs 8: Comparison in drivers of variability	<b>Mission</b> Obs 6: Expertise of human operators Obs 7: Human-robot interaction
<b>Strategies</b> Obs 9: Installation process Obs 10: Scenario modelling Obs 11: Generic configurations	<b>Mechanisms</b> Obs 12: Scenario configuration and parameters Obs 13: Operator-driven map configuration Obs 14: Mechanisms for customers Obs 15: Mechanisms for adaptation rules Obs 16: Contextual navigation	<b>RQ2: Variability management practices</b>  <b>Strategies</b> Obs 17: Community-based resources Obs 18: Collaboration with customers Obs 19: Decoupling and interfaces' harmonization Obs 20: Inter-projects communication Obs 21: Unify codebases & harmonize interfaces  <b>Mechanisms</b> Obs 22: Middleware Obs 23: Certification standards Obs 24: Version control Obs 25: Reuse mechanisms Obs 26: Libraries
	<b>Potential failures</b>  Obs 29: Comparison in variability management	<b>Strategies</b> Obs 27: Generic missions

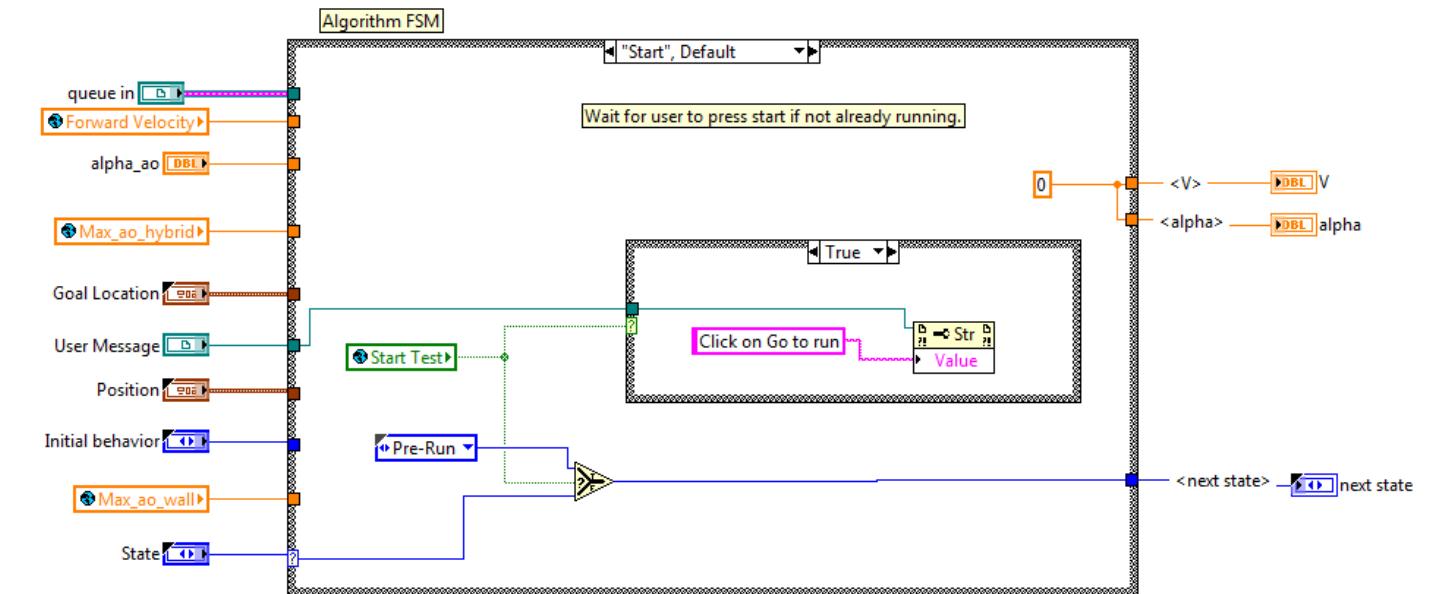
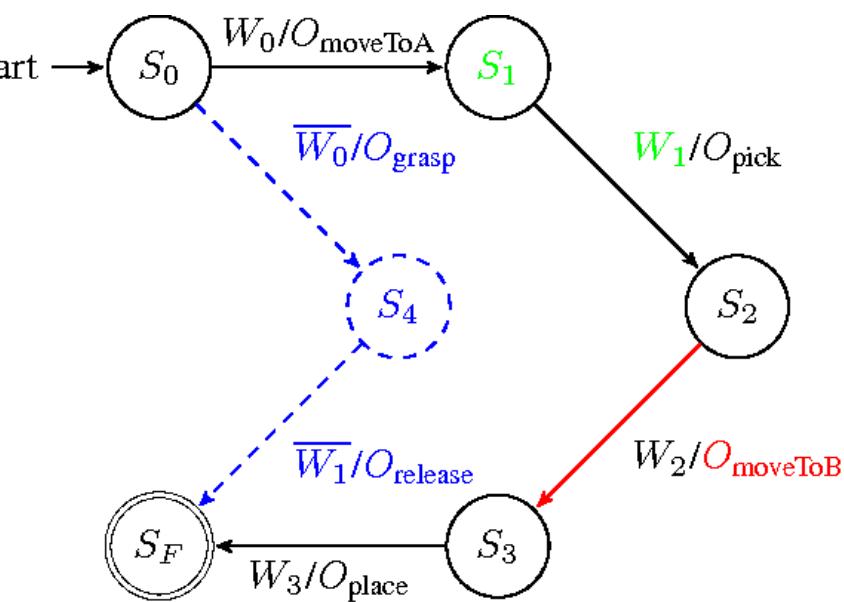
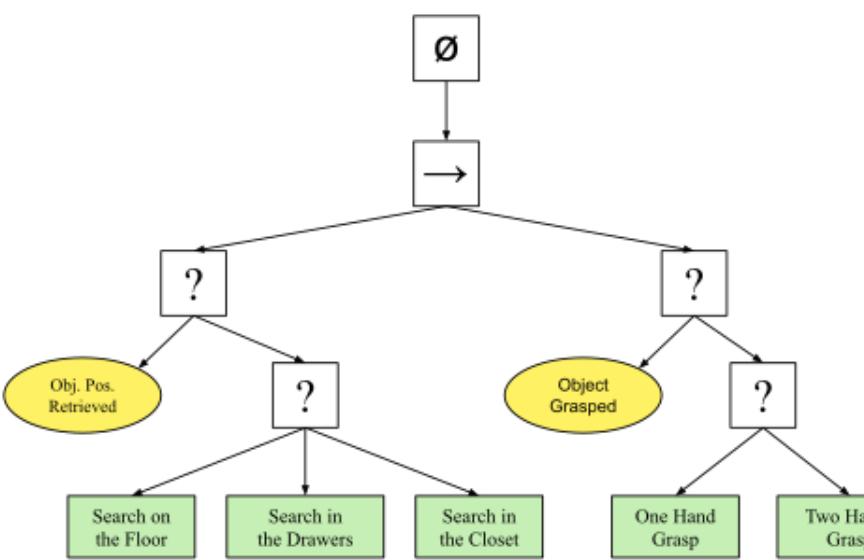
- 
- 1. Democratization**
  - 2. Need of Turn-key solutions**
  - 3. Variability of the real world**

**How to specify missions?**

$$\Phi_1 = \langle \rangle ((r \text{ in } l_1) \text{ && } \langle \rangle (r \text{ in } l_2))$$



Simple enough?



```

14.
15. # Move the robot to the reference point:
16. robot.MoveJ(target)
17.
18. # Draw a hexagon around the reference target:
19. for i in range(7):
20.     ang = i*2*pi/6 #ang = 0, 60, 120, ..., 360
21.
22.     # Calculate the new position around the reference:
23.     x = xyz_ref[0] + R*cos(ang) # new X coordinate
24.     y = xyz_ref[1] + R*sin(ang) # new Y coordinate
25.     z = xyz_ref[2] # new Z coordinate
26.     target_pos.setPos([x,y,z])
27.
28.     # Move to the new target:
29.     robot.MoveL(target_pos)
30.
  
```

# Means to specify robotic missions

Temporal logic:  $\Phi_1 = <>((r \text{ in } l_1) \And <>(r \text{ in } l_2))$

## Logic-based specification of robotic missions

- **Pros:**
  - Clear semantics and unambiguous specification
  - Can be directly used by planners to generate plans or synthesis approaches to generate controllers
  - Enable automatic verification
- **Cons:**
  - Require specific competencies and error-prone
  - Impossible or difficult to specify missions that are complex and with high variability

# Logic-based specification of robotic missions

## The logic of bugs

**Author:**  Gerard J. Holzmann

[Authors Info & Claims](#)

SIGSOFT '02/FSE-10: Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering  
Pages 81 - 87 • <https://doi.org/10.1145/587051.587064>

**Published:** 18 November 2002 [Publication History](#)

 Check for updates

” 51 ↗ 738



### Abstract

Real-life bugs are successful because of their unfailing ability to adapt. In particular this applies to their ability to adapt to strategies that are meant to eradicate them as a species. Software bugs have some of these same traits. We will discuss these traits, and consider what we can do about them.

How to make logic  
more accessible  
and user friendly?

Let's take  
inspiration from  
the formal  
verification world

## Problem space

- Temporal Properties are typically specified as formulae in suitable temporal logics
- The inherent complexity of Temporal Logic formulae may induce to specify properties in a wrong way

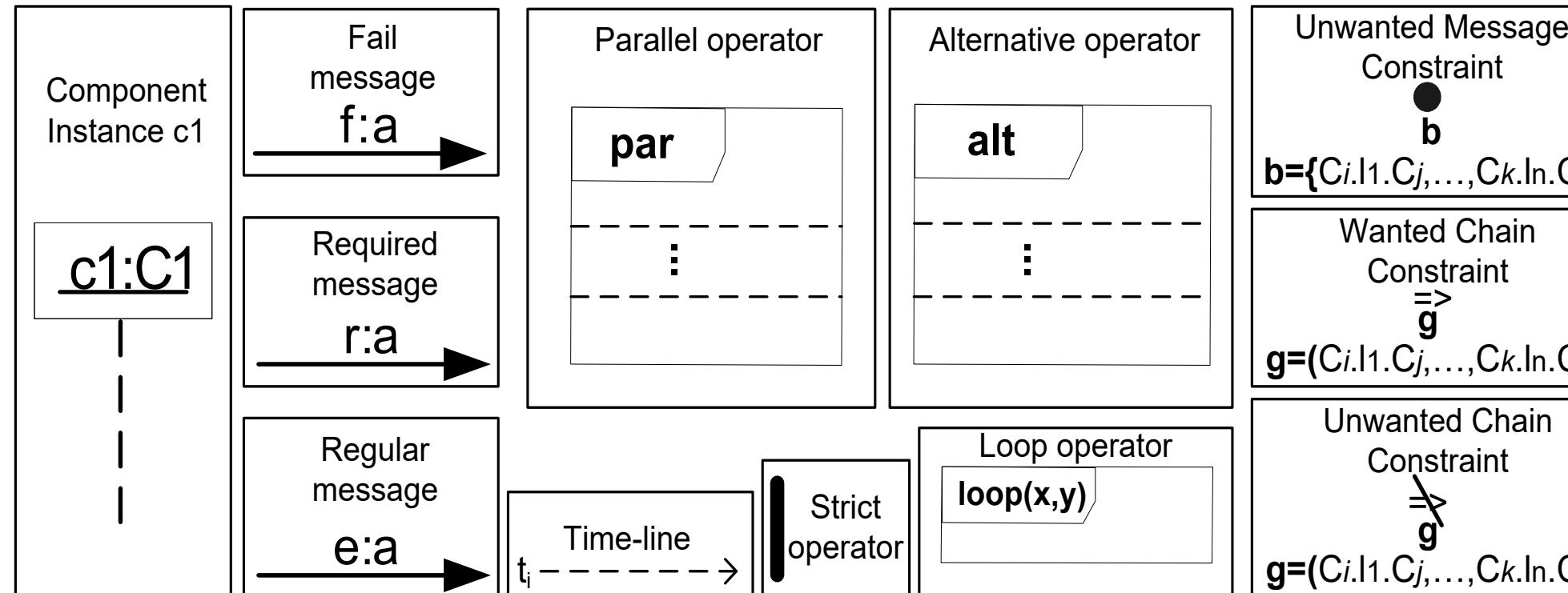
## Solution space

- Languages to facilitate the temporal properties specification
- Property Specification Patterns

# Main idea

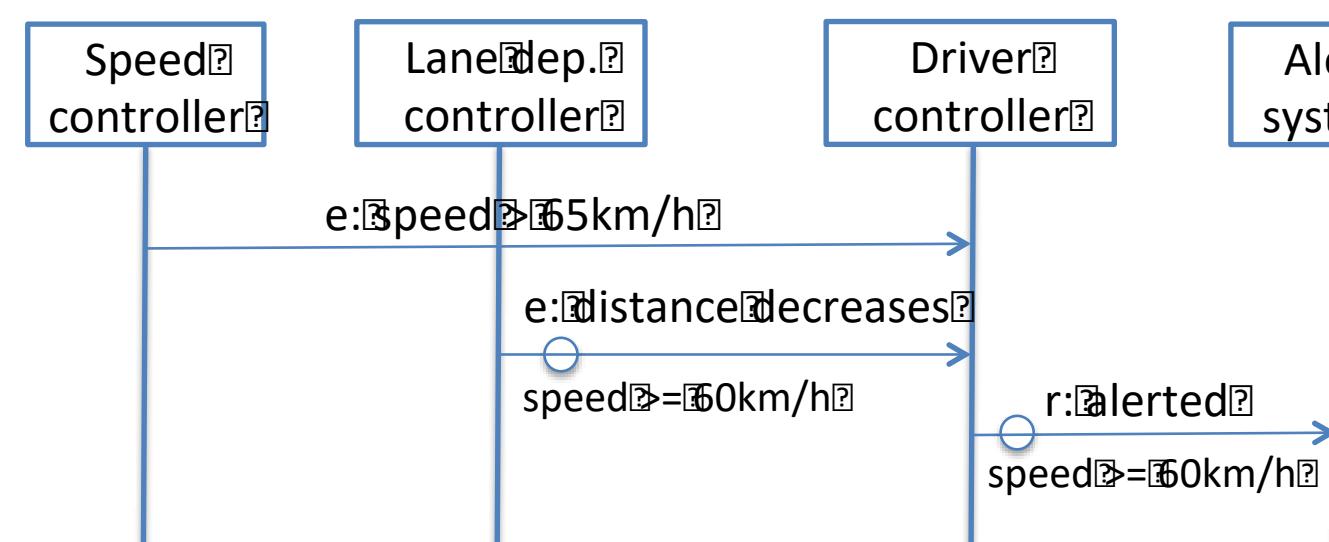
Reduce the expressivity to what is really  
needed and simplify

# Properties Sequence Chart (PSC)



- **Extensions and uses of PSC**

- **Timed Property Sequence Chart (TPSC) -** <http://dx.doi.org/10.1016/j.jss.2009.09.013>
- **Probabilistic Timed Property Sequence Chart (PTPSC) -** <http://dx.doi.org/10.1109/ASE.2009.56>
- **Monitoring of PSC and TPSC properties -** [http://dx.doi.org/10.1007/978-3-642-16612-9\\_39](http://dx.doi.org/10.1007/978-3-642-16612-9_39)
- **Monitoring of PTPSC -** <http://onlinelibrary.wiley.com/doi/10.1002/spe.1038/abstract>



PSC is one of the notations adopted within the Presto project (ARTEMIS-2010-1-269362)

<http://www.presto-embedded.eu/>



PSC is the notation used by MSC Tracer to express temporal properties

<http://www.pragmudev.com/product/tracing.html>



PSC is the notation used by SDL-RT V2.3 standard to express temporal properties

<http://www.sdl-rt.org/>

# Property specification patterns

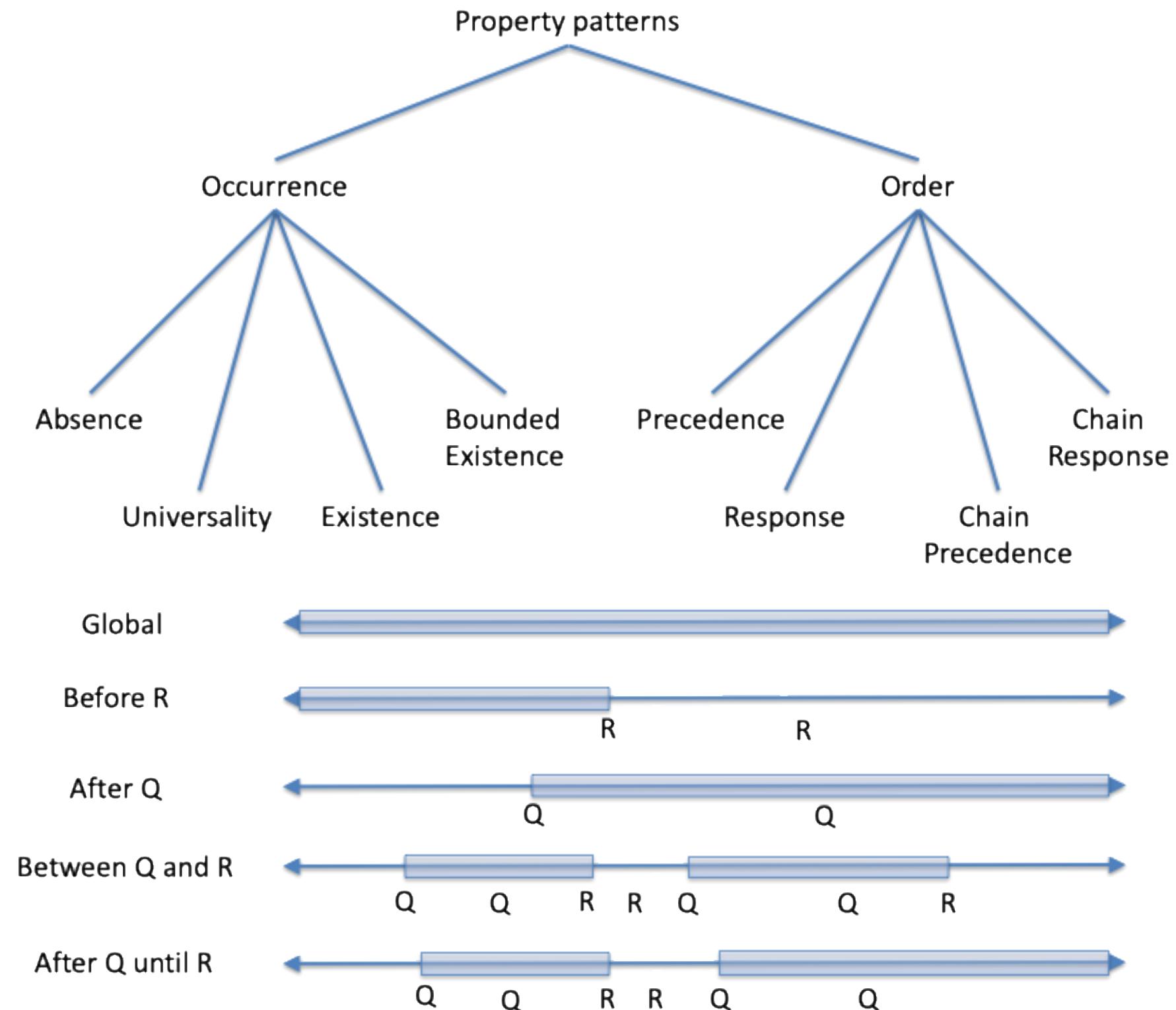
## An example: Response pattern

To describe cause-effect relationships between a pair of events/states. An occurrence of the first, the cause, must be followed by an occurrence of the second, the effect.

Also known as **Follows** and **Leads-to**.

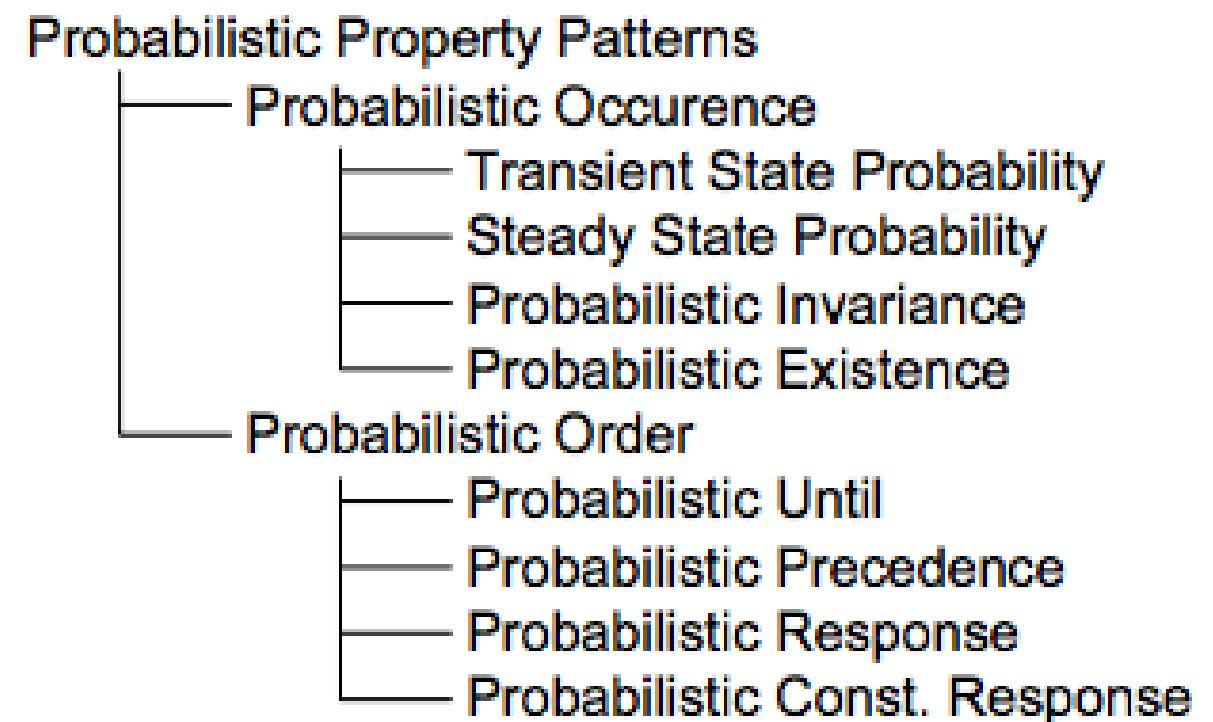
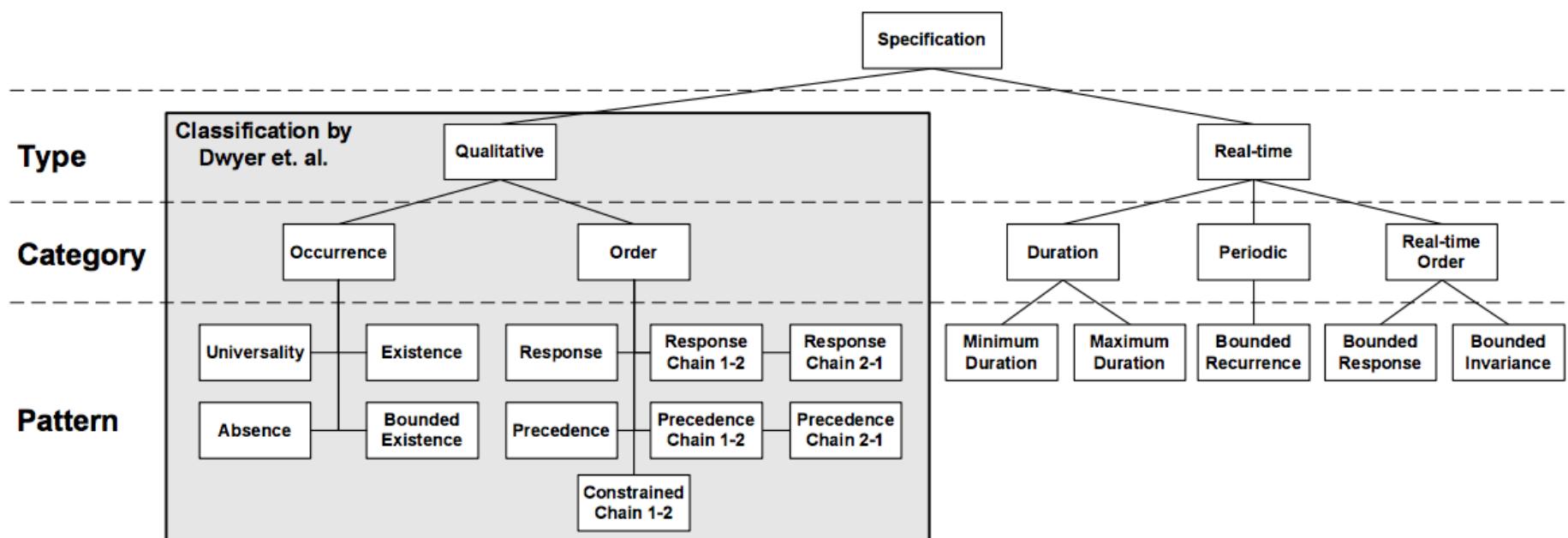
$s$  responds to  $P$  :

Globally	$\square(P \rightarrow \diamond s)$
(*) Before R	$\diamond R \rightarrow (P \rightarrow (\neg R \cup (s \wedge \neg R)) \cup R)$
After Q	$\square(Q \rightarrow \square(P \rightarrow \diamond s))$
(*) Between Q and R	$\square((Q \wedge \neg R \wedge \neg \diamond R) \rightarrow (P \rightarrow (\neg R \cup (s \wedge \neg R)) \cup R))$
(*) After Q until R	$\square(Q \wedge \neg R \rightarrow ((P \rightarrow (\neg R \cup (s \wedge \neg R)) \wedge R)))$



# Real-time specification patterns

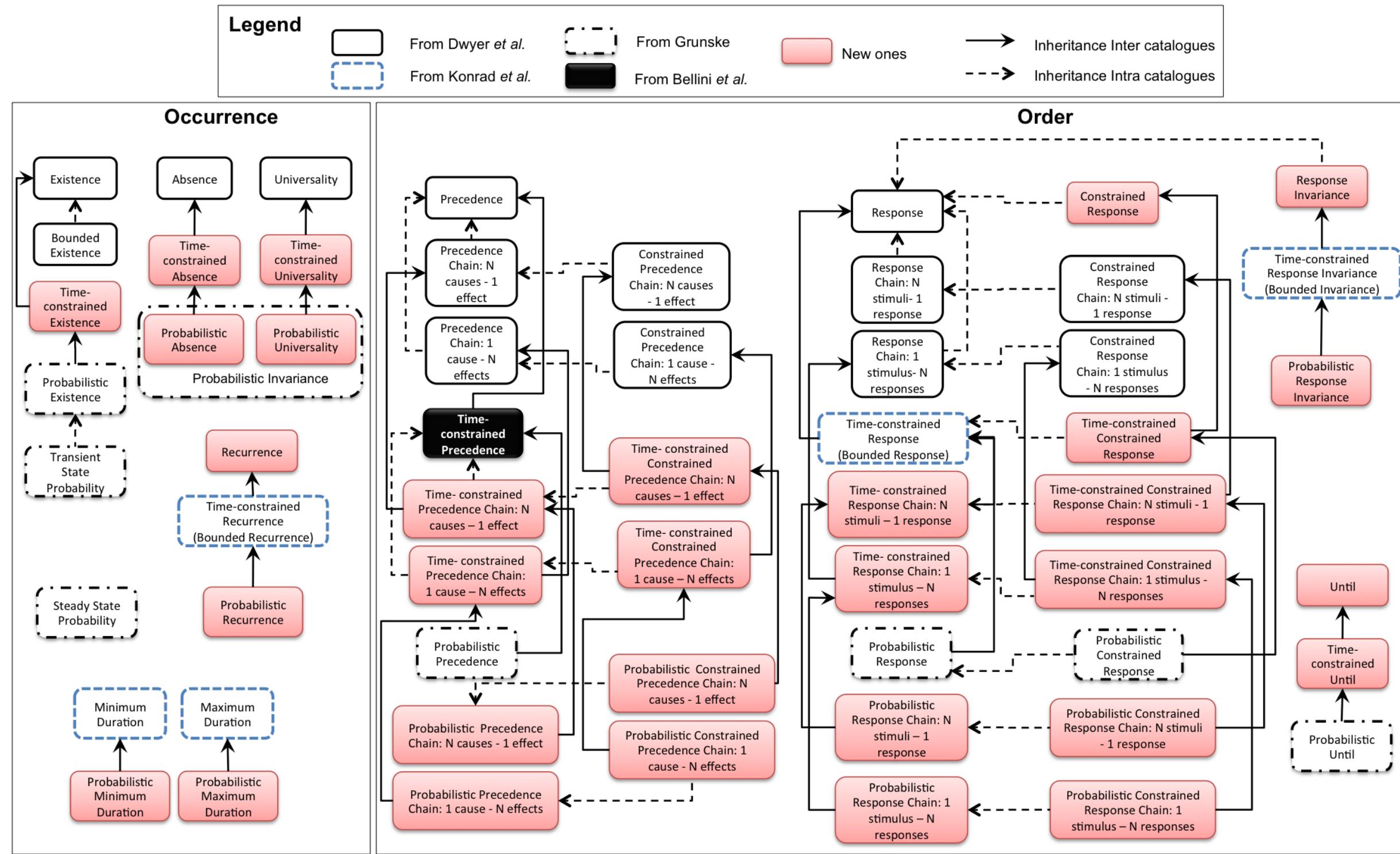
# Probabilistic Property patterns



Sascha Konrad and Betty H. C. Cheng. 2005. Real-time specification patterns. In *Proceedings of the 27th international conference on Software engineering (ICSE '05)*. ACM, New York, NY, USA, 372-381.

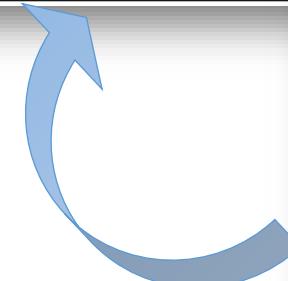
Lars Grunske. 2008. Specification patterns for probabilistic quality properties. In *Proceedings of the 30th international conference on Software engineering (ICSE '08)*. ACM, New York, NY, USA, 31-40.

# Unified catalogue of Property specification patterns

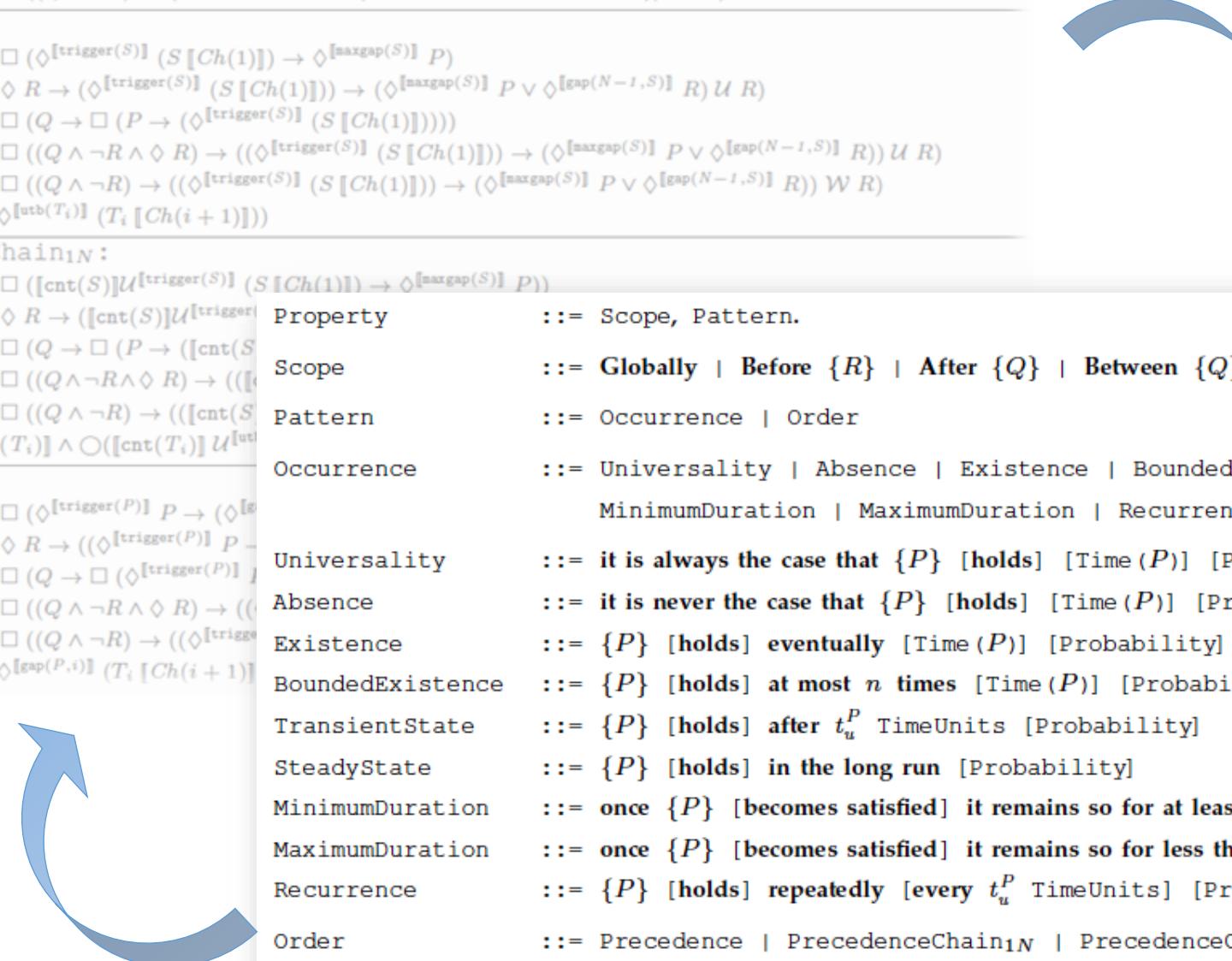


Integration of existing catalogues +  
40 newly identified or extended patterns

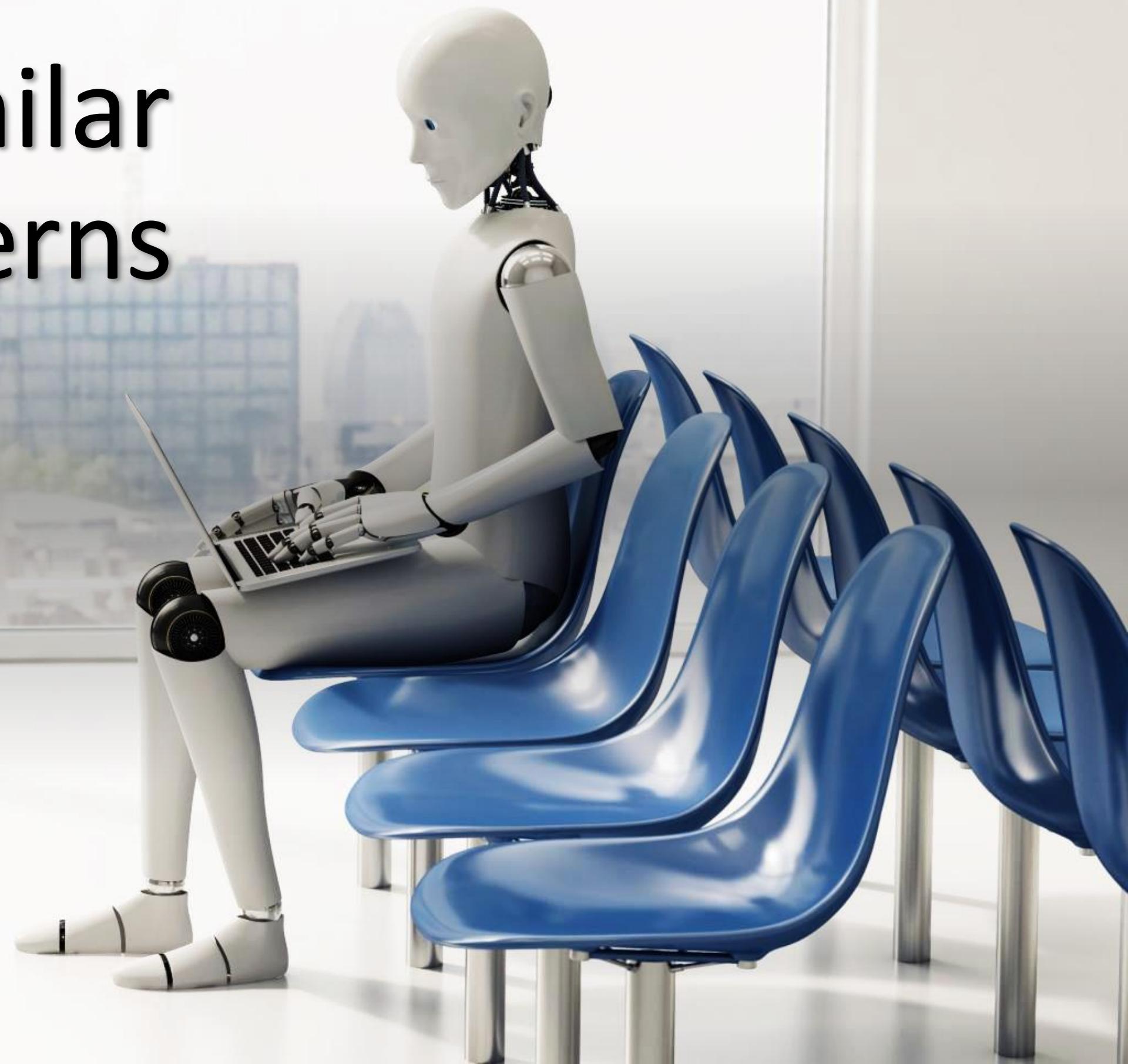
# Property Specification Patterns and Structured English grammar

<b>Precedence:</b> Globally :::= $\square (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow \Diamond \llbracket \text{gap}(P) \rrbracket S)$ Before {R} :::= $\Diamond R \rightarrow (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S \vee \Diamond \llbracket \text{elapsed}(P) \rrbracket R)) \mathcal{U} R$ After {Q} :::= $\square (Q \rightarrow \square (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow \Diamond \llbracket \text{gap}(P) \rrbracket S))$ Between {Q} and {R} :::= $\square ((Q \wedge \neg R \wedge \Diamond R) \rightarrow (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S \vee \Diamond \llbracket \text{elapsed}(P) \rrbracket R)) \mathcal{U} R)$ After {Q} until {R} :::= $\square ((Q \wedge \neg R) \rightarrow (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S \vee \Diamond \llbracket \text{elapsed}(P) \rrbracket R)) \mathcal{W} R)$		<b>PrecedenceChain<sub>1N</sub>:</b> Globally :::= $\square (\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket) \rightarrow \Diamond \llbracket \text{maxgap}(S) \rrbracket P)$ Before {R} :::= $\Diamond R \rightarrow (\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket) \rightarrow (\Diamond \llbracket \text{maxgap}(S) \rrbracket P \vee \Diamond \llbracket \text{gap}(N-1, S) \rrbracket R) \mathcal{U} R)$ After {Q} :::= $\square (Q \rightarrow \square (P \rightarrow (\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket))))$ Between {Q} and {R} :::= $\square ((Q \wedge \neg R \wedge \Diamond R) \rightarrow ((\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket)) \rightarrow (\Diamond \llbracket \text{maxgap}(S) \rrbracket P \vee \Diamond \llbracket \text{gap}(N-1, S) \rrbracket R)) \mathcal{U} R)$ After {Q} until {R} :::= $\square ((Q \wedge \neg R) \rightarrow ((\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket)) \rightarrow (\Diamond \llbracket \text{maxgap}(S) \rrbracket P \vee \Diamond \llbracket \text{gap}(N-1, S) \rrbracket R)) \mathcal{W} R)$ with $\llbracket Ch(i) \rrbracket = \wedge \bigcirc (\Diamond \llbracket \text{utb}(T_i) \rrbracket (T_i \llbracket Ch(i+1) \rrbracket))$
<b>ConstrainedPrecedenceChain<sub>1N</sub>:</b> Globally :::= $\square ([\text{cnt}(S)] \mathcal{U} \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket) \rightarrow \square [\text{Property}]$ Before {R} :::= $\Diamond R \rightarrow ([\text{cnt}(S)] \mathcal{U} \llbracket \text{trigger}(S) \rrbracket$ After {Q} :::= $\square (Q \rightarrow \square (P \rightarrow ([\text{cnt}(S)] \mathcal{U}$ Between {Q} and {R} :::= $\square ((Q \wedge \neg R \wedge \Diamond R) \rightarrow (([\text{cnt}(S)] \mathcal{U}$ After {Q} until {R} :::= $\square ((Q \wedge \neg R) \rightarrow (([\text{cnt}(S)] \mathcal{U}$ with $\llbracket Ch(i) \rrbracket = \wedge \bigcirc ([\text{cnt}(T_i)] \wedge \bigcirc ([\text{cnt}(T_i)] \mathcal{U} \llbracket \text{utb}(T_i) \rrbracket$		<b>Property</b> :::= Scope, Pattern. <b>Scope</b> :::= Globally   Before {R}   After {Q}   Between {Q} and {R}   After {Q} until {R} <b>Pattern</b> :::= Occurrence   Order <b>Occurrence</b> :::= Universality   Absence   Existence   BoundedExistence   TransientState   SteadyState   MinimumDuration   MaximumDuration   Recurrence <b>Universality</b> :::= it is always the case that {P} [holds] [Time(P)] [Probability] <b>Absence</b> :::= it is never the case that {P} [holds] [Time(P)] [Probability] <b>Existence</b> :::= {P} [holds] eventually [Time(P)] [Probability] <b>BoundedExistence</b> :::= {P} [holds] at most n times [Time(P)] [Probability] <b>TransientState</b> :::= {P} [holds] after $t_u^P$ TimeUnits [Probability] <b>SteadyState</b> :::= {P} [holds] in the long run [Probability] <b>MinimumDuration</b> :::= once {P} [becomes satisfied] it remains so for at least $t_u^P$ TimeUnits [Probability] <b>MaximumDuration</b> :::= once {P} [becomes satisfied] it remains so for less than $t_u^P$ TimeUnits [Probability] <b>Recurrence</b> :::= {P} [holds] repeatedly [every $t_u^P$ TimeUnits] [Probability] <b>Order</b> :::= Precedence   PrecedenceChain <sub>1N</sub>   PrecedenceChain <sub>N1</sub>   Until   Response   ResponseChain <sub>1N</sub>   ResponseChain <sub>N1</sub>   ResponseInvariance <b>Precedence</b> :::= if {P} [holds] then it must have been the case that {S} [has occurred] [Interval(P)] before {P} [holds] [Probability] <b>PrecedenceChain<sub>1N</sub></b> :::= if {S} [has occurred] and afterwards ({T <sub>i</sub> } [UpperTimeBound(T <sub>i</sub> )] [Constraint(T <sub>i</sub> )]) <sup>(1 ≤ i ≤ N-1; “,”)</sup> [hold] then it must have been the case that {P} [has occurred] [Interval(S)] before {S} [holds] [Constraint(S)] [Probability]

# Property Specification Patterns and Structured English grammar

Precedence:	
Globally	$\square (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow \Diamond \llbracket \text{gap}(P) \rrbracket S)$
Before $\{R\}$	$\Diamond R \rightarrow (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S \vee \Diamond \llbracket \text{elapsed}(P) \rrbracket R)) \cup R$
After $\{Q\}$	$\square (Q \rightarrow \square (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow \Diamond \llbracket \text{gap}(P) \rrbracket S))$
Between $\{Q\}$ and $\{R\}$	$\square ((Q \wedge \neg R \wedge \Diamond R) \rightarrow (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S \vee \Diamond \llbracket \text{elapsed}(P) \rrbracket R)) \cup R)$
After $\{Q\}$ until $\{R\}$	$\square ((Q \wedge \neg R) \rightarrow (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S \vee \Diamond \llbracket \text{elapsed}(P) \rrbracket R)) \mathcal{W} R)$
PrecedenceChain <sub>1N</sub> :	
Globally	$\square (\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket) \rightarrow \Diamond \llbracket \text{maxgap}(S) \rrbracket P)$
Before $\{R\}$	$\Diamond R \rightarrow (\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket)) \rightarrow (\Diamond \llbracket \text{maxgap}(S) \rrbracket P \vee \Diamond \llbracket \text{gap}(N-1, S) \rrbracket R) \cup R$
After $\{Q\}$	$\square (Q \rightarrow \square (P \rightarrow (\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket)))$
Between $\{Q\}$ and $\{R\}$	$\square ((Q \wedge \neg R \wedge \Diamond R) \rightarrow ((\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket)) \rightarrow (\Diamond \llbracket \text{maxgap}(S) \rrbracket P \vee \Diamond \llbracket \text{gap}(N-1, S) \rrbracket R)) \cup R)$
After $\{Q\}$ until $\{R\}$	$\square ((Q \wedge \neg R) \rightarrow ((\Diamond \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket)) \rightarrow (\Diamond \llbracket \text{maxgap}(S) \rrbracket P \vee \Diamond \llbracket \text{gap}(N-1, S) \rrbracket R)) \mathcal{W} R)$
with $\llbracket Ch(i) \rrbracket = \wedge \bigcirc (\Diamond \llbracket \text{utb}(T_i) \rrbracket (T_i \llbracket Ch(i+1) \rrbracket))$	
ConstrainedPrecedenceChain <sub>1N</sub> :	
Globally	$\square ([\text{cnt}(S)] \mathcal{U} \llbracket \text{trigger}(S) \rrbracket (S \llbracket Ch(1) \rrbracket) \rightarrow \Diamond \llbracket \text{maxgap}(S) \rrbracket P)$
Before $\{R\}$	$\Diamond R \rightarrow ([\text{cnt}(S)] \mathcal{U} \llbracket \text{trigger}(S) \rrbracket P)$
After $\{Q\}$	$\square (Q \rightarrow \square (P \rightarrow ([\text{cnt}(S)] \mathcal{U} \llbracket \text{trigger}(S) \rrbracket P)))$
Between $\{Q\}$ and $\{R\}$	$\square ((Q \wedge \neg R \wedge \Diamond R) \rightarrow (([\text{cnt}(S)] \mathcal{U} \llbracket \text{trigger}(S) \rrbracket P) \vee ([\text{cnt}(S)] \mathcal{U} \llbracket \text{trigger}(S) \rrbracket R)))$
After $\{Q\}$ until $\{R\}$	$\square ((Q \wedge \neg R) \rightarrow (([\text{cnt}(S)] \mathcal{U} \llbracket \text{trigger}(S) \rrbracket P) \vee ([\text{cnt}(S)] \mathcal{U} \llbracket \text{trigger}(S) \rrbracket R)))$
with $\llbracket Ch(i) \rrbracket = \wedge \bigcirc ([\text{cnt}(T_i)] \wedge \bigcirc ([\text{cnt}(T_i)] \mathcal{U} \llbracket \text{trigger}(T_i) \rrbracket P))$	
PrecedenceChain <sub>N1</sub> :	
Globally	$\square (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S))$
Before $\{R\}$	$\Diamond R \rightarrow ((\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S)) \vee (\Diamond \llbracket \text{elapsed}(P) \rrbracket R)) \cup R$
After $\{Q\}$	$\square (Q \rightarrow \square (\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S)))$
Between $\{Q\}$ and $\{R\}$	$\square ((Q \wedge \neg R \wedge \Diamond R) \rightarrow ((\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S)) \vee ((\Diamond \llbracket \text{elapsed}(P) \rrbracket R \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S)))) \cup R)$
After $\{Q\}$ until $\{R\}$	$\square ((Q \wedge \neg R) \rightarrow ((\Diamond \llbracket \text{trigger}(P) \rrbracket P \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S)) \vee ((\Diamond \llbracket \text{elapsed}(P) \rrbracket R \rightarrow (\Diamond \llbracket \text{gap}(P) \rrbracket S)))) \mathcal{W} R)$
with $\llbracket Ch(i) \rrbracket = \wedge \bigcirc (\Diamond \llbracket \text{gap}(P,i) \rrbracket (T_i \llbracket Ch(i+1) \rrbracket))$	
	
<b>Property</b> ::= Scope, Pattern. <b>Scope</b> ::= Globally   Before {R}   After {Q}   Between {Q} and {R}   After {Q} until {R} <b>Pattern</b> ::= Occurrence   Order <b>Occurrence</b> ::= Universality   Absence   Existence   BoundedExistence   TransientState   SteadyState   MinimumDuration   MaximumDuration   Recurrence <b>Universality</b> ::= it is always the case that {P} [holds] [Time(P)] [Probability] <b>Absence</b> ::= it is never the case that {P} [holds] [Time(P)] [Probability] <b>Existence</b> ::= {P} [holds] eventually [Time(P)] [Probability] <b>BoundedExistence</b> ::= {P} [holds] at most n times [Time(P)] [Probability] <b>TransientState</b> ::= {P} [holds] after $t_u^P$ TimeUnits [Probability] <b>SteadyState</b> ::= {P} [holds] in the long run [Probability] <b>MinimumDuration</b> ::= once {P} [becomes satisfied] it remains so for at least $t_u^P$ TimeUnits [Probability] <b>MaximumDuration</b> ::= once {P} [becomes satisfied] it remains so for less than $t_u^P$ TimeUnits [Probability] <b>Recurrence</b> ::= {P} [holds] repeatedly [every $t_u^P$ TimeUnits] [Probability] <b>Order</b> ::= Precedence   PrecedenceChain <sub>1N</sub>   PrecedenceChain <sub>N1</sub>   Until   Response   ResponseChain <sub>1N</sub>   ResponseChain <sub>N1</sub>   ResponseInvariance <b>Precedence</b> ::= if {P} [holds] then it must have been the case that {S} [has occurred] [Interval(P)] before {P} [holds] [Probability] <b>PrecedenceChain<sub>1N</sub></b> ::= if {S} [has occurred] and afterwards ({T <sub>i</sub> } [UpperTimeBound(T <sub>i</sub> )] [Constraint(T <sub>i</sub> )]) <sup>(1 ≤ i ≤ N-1; ",")</sup> [hold] then it must have been the case that {P} [has occurred] [Interval(S)] before {S} [holds] [Constraint(S)] [Probability]	

Can we define similar  
specification patterns  
for robots?





# Let's first discuss another important aspect

Simplicity but keeping rigorousness

In this context, Ambiguity is evil!

Intuitive and Simple **but also** Rigorous

“A robot  $r$  shall visit the two locations  $l_1$  and  $l_2$  in this order”

# Intuitive and Simple **but also** Rigorous

“A robot  $r$  shall visit the two locations  $l_1$  and  $l_2$  in this order”

$l_1$  and then  $l_2$

# Intuitive and Simple **but also** Rigorous

Ambiguity

“A robot r shall visit the two locations l1 and l2 in this order”

l1 and then l2

Is it possible to visit l2 before l1 and then to visit l2?

# Intuitive and Simple **but also** Rigorous

“A robot r shall visit the two locations l1 and l2 in this order”

|l1 and then l2

Is it possible to visit l2 before l1 and then to visit l2?

$\phi_1 = \langle\langle (r \text{ in } l1) \& \& \langle\langle (r \text{ in } l2))$

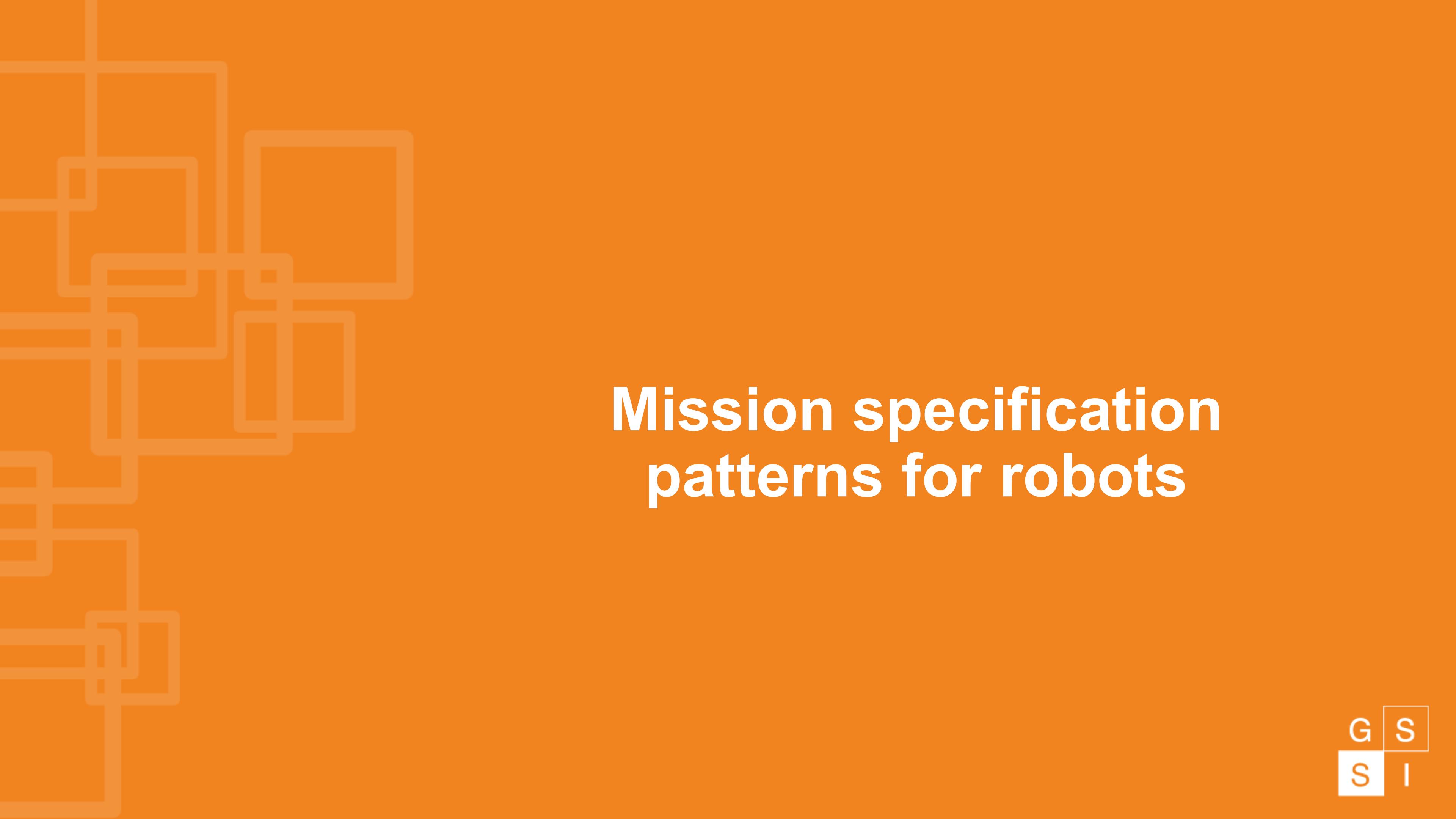
VS.

$\phi_2 = \phi_1 \& \& ((\neg r \text{ in } l2) \cup (r \text{ in } l1))$

# Intuitive and Simple **but also** Rigorous

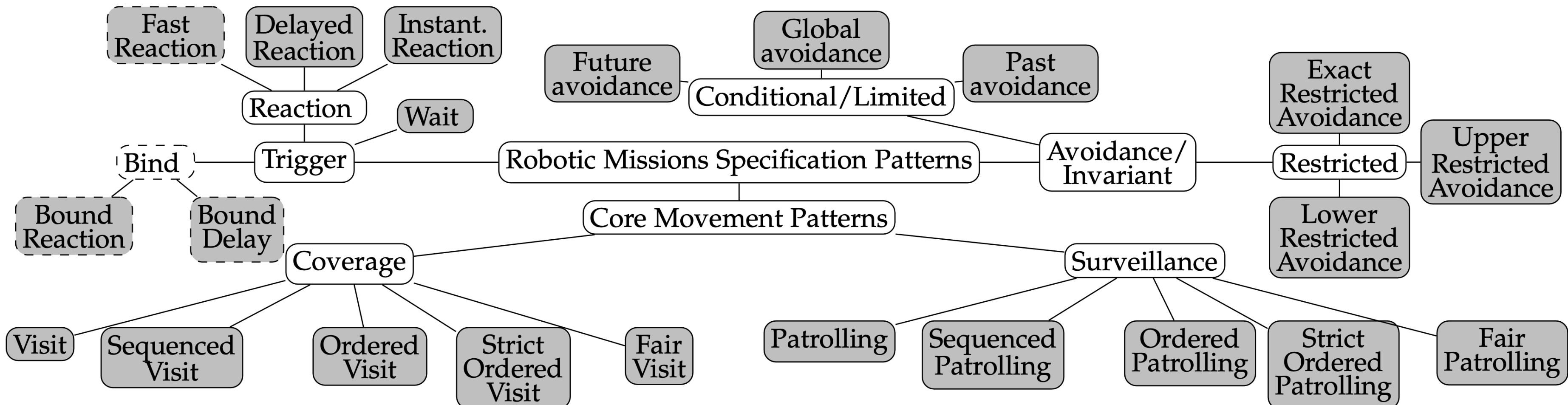
If we allow an ambiguous specification,  
which behavior will have the robot,  
and who will decide it?

**This is why ambiguity is evil!**



# Mission specification patterns for robots

# Specification Patterns



# Specification Patterns

	Description	Example	Formula ( $l_1, l_2, \dots$ are location propositions)
Visit	Visit a set of locations in an unspecified order.	Locations $l_1, l_2$ , and $l_3$ must be visited. $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_{\#})^\omega$ is an example trace that satisfies the mission requirement.	$\bigwedge_{i=1}^n \mathcal{F}(l_i)$
Sequenced Visit	Visit a set of locations in sequence, one after the other.	Locations $l_1, l_2, l_3$ must be covered following this sequence. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_{\#\setminus 3})^\omega$ violates the mission since $l_3$ does not follow $l_2$ . The trace $l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_{\#})^\omega$ satisfies the mission requirement.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \dots \mathcal{F}(l_n)))$
Ordered Visit	The sequenced visit pattern does not forbid to visit a successor location before its predecessor, but only that after the predecessor is visited the successor is also visited. Ordered visit forbids a successor to be visited before its predecessor.	Locations $l_1, l_2, l_3$ must be covered following this order. The trace $l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow l_3 \rightarrow (l_{\#})^\omega$ does not satisfy the mission requirement since $l_3$ precedes $l_2$ . The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_{\#})^\omega$ satisfies the mission requirement.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \dots \mathcal{F}(l_n)))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1}) \mathcal{U} l_i$
Strict Ordered Visit	The ordered visit pattern does not avoid a predecessor location to be visited multiple times before its successor. Strict ordered visit forbids this behavior.	Locations $l_1, l_2, l_3$ must be covered following the strict order $l_1, l_2, l_3$ . The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_{\#})^\omega$ does not satisfy the mission requirement since $l_1$ occurs twice before $l_2$ . The trace $l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow l_4 \rightarrow l_3 \rightarrow (l_{\#})^\omega$ satisfies the mission requirement.	$\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \dots \mathcal{F}(l_n)))$ $\bigwedge_{i=1}^{n-1} (\neg l_{i+1}) \mathcal{U} l_i$ $\bigwedge_{i=1}^{n-1} (\neg l_i) U(l_i \wedge \mathcal{X}(\neg l_i \mathcal{U}(l_{i+1})))$
Fair Visit	The difference among the number of times locations within a set are visited is at most one.	Locations $l_1, l_2, l_3$ must be covered in a fair way. The trace $l_1 \rightarrow l_4 \rightarrow l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow (l_{\#\setminus\{1,2,3\}})^\omega$ does not perform a fair visit since it visits $l_1$ three times while $l_2$ and $l_3$ are visited once. The trace $l_1 \rightarrow l_4 \rightarrow l_3 \rightarrow l_1 \rightarrow l_4 \rightarrow l_2 \rightarrow l_2 \rightarrow l_4 \rightarrow (l_{\#\setminus\{1,2,3\}})^\omega$ performs a fair visit since it visits locations $l_1, l_2$ , and $l_3$ twice.	$\bigwedge_{i=1}^n \mathcal{F}(l_i)$ $\bigwedge_{i=1}^n \mathcal{G}(l_i \rightarrow \mathcal{X}((\neg l_i) \mathcal{W} l_{(i+1)\%n}))$

# An example of pattern

**Name:** Strict Ordered Patrolling

**Intent:** A robot must patrol a set of locations following a strict sequence ordering. Such locations can be, e.g., areas in a building to be surveyed.

**Template:** The following formula encodes the mission in LTL for  $n$  locations and a robot  $r$  ( $\%$  is the modulo arithmetic operator):

$$\bigwedge_{i=1}^n \mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2 \wedge \dots \mathcal{F}(l_n)))) \bigwedge_{i=1}^{n-1} ((\neg l_{i+1}) \mathcal{U} l_i) \bigwedge_{i=1}^n \mathcal{G}(l_{(i+1)\%n} \rightarrow \mathcal{X}((\neg l_{(i+1)\%n}) \mathcal{U} l_i))$$

Example with two locations.

$$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2))) \wedge ((\neg l_2) \mathcal{U} l_1) \wedge \mathcal{G}(l_2 \rightarrow \mathcal{X}((\neg l_2) \mathcal{U} l_1)) \wedge \mathcal{G}(l_1 \rightarrow \mathcal{X}((\neg l_1) \mathcal{U} l_2))$$

where  $l_1$  and  $l_2$  are expressions that indicate that a robot  $r$  is in locations  $l_1$  and  $l_2$ , respectively.

**Variations:** A developer may want to allow traces in which sequences of *consecutive*  $l_1$  ( $l_2$ ) are allowed, that is strict ordering is applied on sequences of non consecutive  $l_1$  ( $l_2$ ). In this case, traces in the form  $l_1 \rightarrow (\rightarrow l_1 \rightarrow l_1 \rightarrow l_3 \rightarrow l_2)^\omega$  are admitted, while traces in the form  $l_1 \rightarrow (\rightarrow l_1 \rightarrow l_3 \rightarrow l_1 \rightarrow l_2)^\omega$  are not admitted. This variation can be encoded using the following specification:

$$\mathcal{G}(\mathcal{F}(l_1 \wedge \mathcal{F}(l_2))) \wedge ((\neg l_2) \mathcal{U} l_1) \wedge \mathcal{G}((l_2 \wedge \mathcal{X}(\neg l_2)) \rightarrow \mathcal{X}((\neg l_2) \mathcal{U} l_1)) \wedge \mathcal{G}((l_1 \wedge \mathcal{X}(\neg l_1)) \rightarrow \mathcal{X}((\neg l_1) \mathcal{U} l_2))$$

This specification allows for sequences of consecutive  $l_1$  ( $l_2$ ) since the left side of the implication  $l_1 \wedge \mathcal{X}(\neg l_1)$  ( $l_2 \wedge \mathcal{X}(\neg l_2)$ ) is only triggered when  $l_1$  ( $l_2$ ) is exited.

**Examples and Known Uses:** A common usage example of the Strict Ordered Patrolling pattern is a scenario where a robot is performing surveillance in a building during night hours. Strict Sequence Patrolling and Avoidance often go together. Avoidance patterns are used to force robots to avoid obstacles as they guard a location. Triggers can also be used in combination with the Strict Sequence Patrolling pattern to specify conditions upon which Patrolling should start or stop.

**Relationships:** The Strict Ordered Patrolling pattern is a specialisation of the Ordered Patrolling pattern, forcing the strict ordering.

**Occurrences:** Smith et. al. [74] proposed a mission specification forcing a robot to not visit a location twice in a row before a target location is reached.

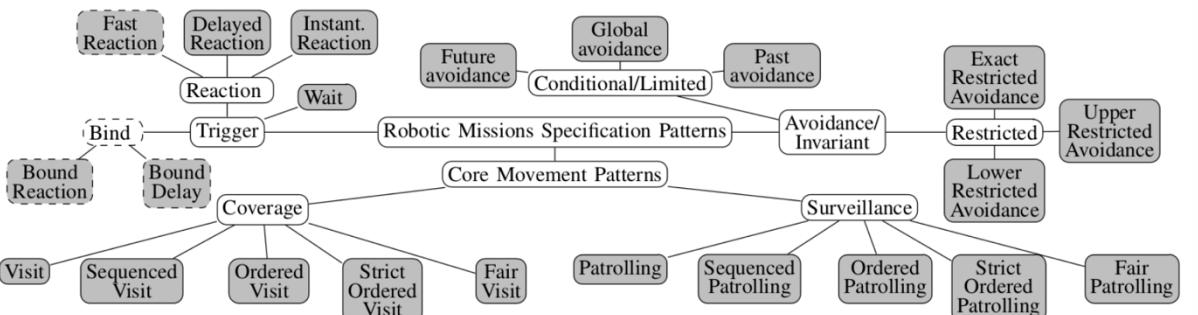
# Further info about specification patterns



This page complements the paper "Specification Patterns for Robotic Missions" and is an online repository of a specification pattern catalog for missions of mobile robots. The pattern system is not intended to be exhaustive or complete, and the repository is not intended to be static. The set of patterns will grow over time as designers specify missions that do not belong to the provided patterns.

You can further find the [patterns](#), information on [evaluation](#), [requirements collection](#) and tool support through [PsALM](#). Reproduction kits, specifications and accompanying code can be found in [experiments](#).

## **Pattern Catalog**



<http://roboticpatterns.com>

Claudio Menghi, Christos Tsigkanos, Patrizio Pelliccione, Carlo Ghezzi, and Thorsten Berger, Specification Patterns for Robotic Missions, **Transactions on Software Engineering (TSE)**, 2019

# Journal First-track at ICSE 2020

Claudio Menghi, Christos Tsigkanos, Thorsten Berger, and Patrizio Pelliccione, PsALM: Specification of Dependable Robotic Missions. IEEE/ACM ICSE Demo, 2019.

<https://github.com/claudiomenghi/PsALM>



**Specification Patterns for Robotic Missions**

Specification Patterns for Robotic Missions

Scipio Menghi, Christos Tsagkias, Patrizio Pellicone, Carlo Ghezzi, and Thorsten Berger

**Abstract**—Mobile and general purpose robots increasingly support our everyday life, requiring dependable robotics control software. Creating such systems requires amounts to implementing their complex behaviors known as missions. Recognizing this need, a large number of mission planning approaches have been proposed. However, they all share the same problem: they lack a formalism able to define specific mission types for synthesis, verification, simulation or guiding implementation. For instance, the logical language LTL is a well-known formalism for specifying temporal properties of systems. However, it is not suitable for defining mission types. In fact, domain-specific languages are usually listed to specify robot behaviors, while logical languages such as LTL are often used by researchers to verify mission specifications. This paper proposes a formalism for mission specification that integrates both approaches, combining, and mapping the patterns to create mission specifications. The pattern-based solution provides for recurrent usage by researchers and engineers, and it is also suitable for reuse by domain experts. The proposed approach is based on the notion of mission specification in temporal logic. Our tool produces specifications expressed in the temporal logic LTL, and LTL can be used to verify mission specifications. The proposed approach is illustrated with two examples: a robot mission for a nuclear power plant and a robot mission for a space station. They are evaluated in terms of 441 real-world mission requirements and 125 mission specifications. Five of these mission specifications are generated by the proposed approach.

**INTRODUCTION**

Mobile robots are increasingly used in complex environments aiming at autonomously reasoning about their surroundings and adapting to environment and technology demands software that can isolate growth [16], [11], [12], [13]. Even though it ends up being used for accomplishing tasks of everyday life by end-users, mobile robotics is a challenging field that requires mathematics, physics, and robotics, making a major software engineering challenge [18]. Indeed, building a domain-specific language for mobile robotics is a challenge in itself, as the domain is highly specific and needs to be well understood by domain experts, where electrical engineers develop low-level hardware components that are used in mobile devices; mechanical engineers develop sensors and actuators; electrical engineers develop low-level controlling and managing their movement; and software engineers develop high-level primitives; a software engineer is responsible for what the robot needs to accomplish [16]. Among the different ways of solving this challenge, in this work, we consider declarative approaches. In this way, we consider declarative specifications as the final outcome the software should accomplish. Declarative approaches have been prominently used in the robotics domain [17]. Precisely, the main challenges in declarative approaches are the automation of precise reasoning and the automation of precise reasoning are among the main challenges in robotics [18]. The main idea behind declarative approaches is that the user should be provided with a notation that is high-level and user-friendly [14], [21]. On the other hand, to enable the user to provide a formal and precise description of what he wants the system to do, he should provide a formal and precise description of what he wants the system to do.

Engineering robotics software typically amounts to expressing the robot's mission in natural language (benignly called mission requirement), and then translating this mission requirement into a formal representation. The latter are often expressed in a domain-specific language, many of which have been proposed over the last decades [19]. In this work, we propose a domain-specific language environment used to model what could be executed by a mobile robot in a specific environment. The languages these languages are typically bound to specific types of robots and domains. For example, the languages used in mobile robot works especially from the robotics domain, advocate the use of formally specified missions in temporal logic [20]. In this work, we propose a domain-specific language environment for mobile robots, which is based on a domain-specific formal language that can be too complex and error-prone for non-experts. This is why we propose a domain-specific language that is generally challenging, as widely recognized in the software engineering literature [21], [22], [23], [24], [25], [26], [27], [28].

Mission requirements are often addressed through precise and unambiguous specifications [36], [37]. These specifications are usually composed of several sentences that the user will visit the kitchen and the office." This can be interpreted as the user wanting the robot to visit both the kitchen and the office. However, this interpretation is wrong. The user should visit the "kitchen" with a specific order and visit the "office" with a specific order. Assume that the correct intended behavior requires that "The kitchen and the office" are visited sequentially. In this case, the user's specification problem [36], [37]. When transforming this specification into a formal representation, the user's specification problem, an expert might come up with the following representation:

$$P = \{r \mid r \in K\} \cup \{r \mid r \in O\},$$

where  $K$  and  $O$  is rigorously that robot  $r$  is in the kitchen and office, respectively, and  $P$  denotes finally. Now, recall that the user's intention was to visit the kitchen and the office sequentially. It is important to highlight

19 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)

# FORTRAN Specification of Dependable Robotic Missions

• Engineering dependable software for mobile robots using linguistic primitives. A central aim in robotics research is the mission specification – a description of what the robot must do. In contrast to other fields, such as among others, to synthesis, verify, modify, simulate or guide the execution of programs, mission specification in robotics is particularly challenging, as engineers need to specify the mission in a language that is meaningful in a logical language – a laboratories and review-proof language.

• The mission specification problem has been recognized as a solution for the problem, that is, how to define the mission in a way that can be easily understood by both humans and computers. It has been recognized as a solution for the problem, that is, how to define the mission in a way that can be easily understood by both humans and computers. This can be done by engineers to create complex mission specifications that can be understood by both humans and computers. PAALM, a hardware supporting the development of mission specifications, is a mission specification language that provides a formal specification language and allows users to define mission specifications in a formal language expressed in LTL and CTL temporal logics to be used in mission planning and model checking, supporting mission design.

• The catalog and PAALM is available on our dedicated [webpage](http://www.cs.tufts.edu/~peter/paalm/).

## 1. INTRODUCTION

Robotics are increasingly used in complex environments at automatically realizing various mission such as navigation, delivering items, or following certain paths. These missions are often complex and require a lot of time and many demands. We can thus see that growth, complexity, and robustness of the mission requirements are the chief behavior of robots [11]. To precisely defining the behaviour, i.e., a declarative specification of the behaviour is required. The mission specifications that can be used for automating processing are among the most important mission specifications. On the one hand, mission should be specified in a way that is high-level and user-friendly [7, 38]. On the other hand, mission specifications should be precise and unambiguous and provide a precise description of what the robot must do. The mission specifications that are popular are a pattern and the PAALM specification. The pattern approach is a popular solution to the specification problems [11, 12]. While precise behavioral specifications in the logical languages reader should be aware that

• PAALM [14] is a specification language and editor program [14]. The program is stand-alone, uses plain English with the syntax rules and semantics of logical languages [11], [12], [16].

• PAALM is a hardware supporting the mission planning [17] to facilitate engineering missions for mobile robots [18]. Each part in the mission is a mission specification, which specifies what the robot should do, such as go to a place, move to another place, or move to a place, and – most importantly – a template mission specification in temporal logic. The latter relies on the mission specification language (MSL) [19]. The MSL (CTL) is the most widely used formal specification language in robotics [20], [21], [22]. The catalog has been produced by an international team of researchers and practitioners from robotics research and industry applications.

• PAALM is a hardware supporting the mission planning [17] to facilitate engineering missions for mobile robots [18]. Each part in the mission is a mission specification, which specifies what the robot should do, such as go to a place, move to another place, or move to a place, and – most importantly – a template mission specification in temporal logic. The latter relies on the mission specification language (MSL) [19]. The MSL (CTL) is the most widely used formal specification language in robotics [20], [21], [22]. The catalog has been produced by an international team of researchers and practitioners from robotics research and industry applications.

• PAALM is a hardware supporting the PAALM (Pattern and Mission Specifier) toolchain, which provides compact support for mission planning and mission execution. PAALM supports specifying a mission requirement through a structured English grammar, which uses patterns as a basic building block for mission planning, mission execution, mission monitoring, and (ii) automatically generating mission specifications from mission requirements. PAALM is a mission specification language with a planner [29], NaMvM [30] (a model checker); and Smarty [31] (an algorithm for research and development). PAALM also includes tools [32] (for mission planning environment). The pattern catalog and the PAALM toolchain support mission planning for robotic systems, which is recognized as an important software engineering discipline.

We are using the following scenario to demonstrate system modeling and mission planning.

*A robot is deployed within a university building to deliver coffee to employees. Specifically, the robot picks up coffee from a coffee machine in the lobby and then delivers the coffee to an employee.*

The mission specifications involved, various functionalities of PAALM as well as real robot execution of the resulting mission.

99  
9/19/\$1.00 ©2019 IEEE  
9/ICSE-Companion.2019.00048

# Are these patterns enough?

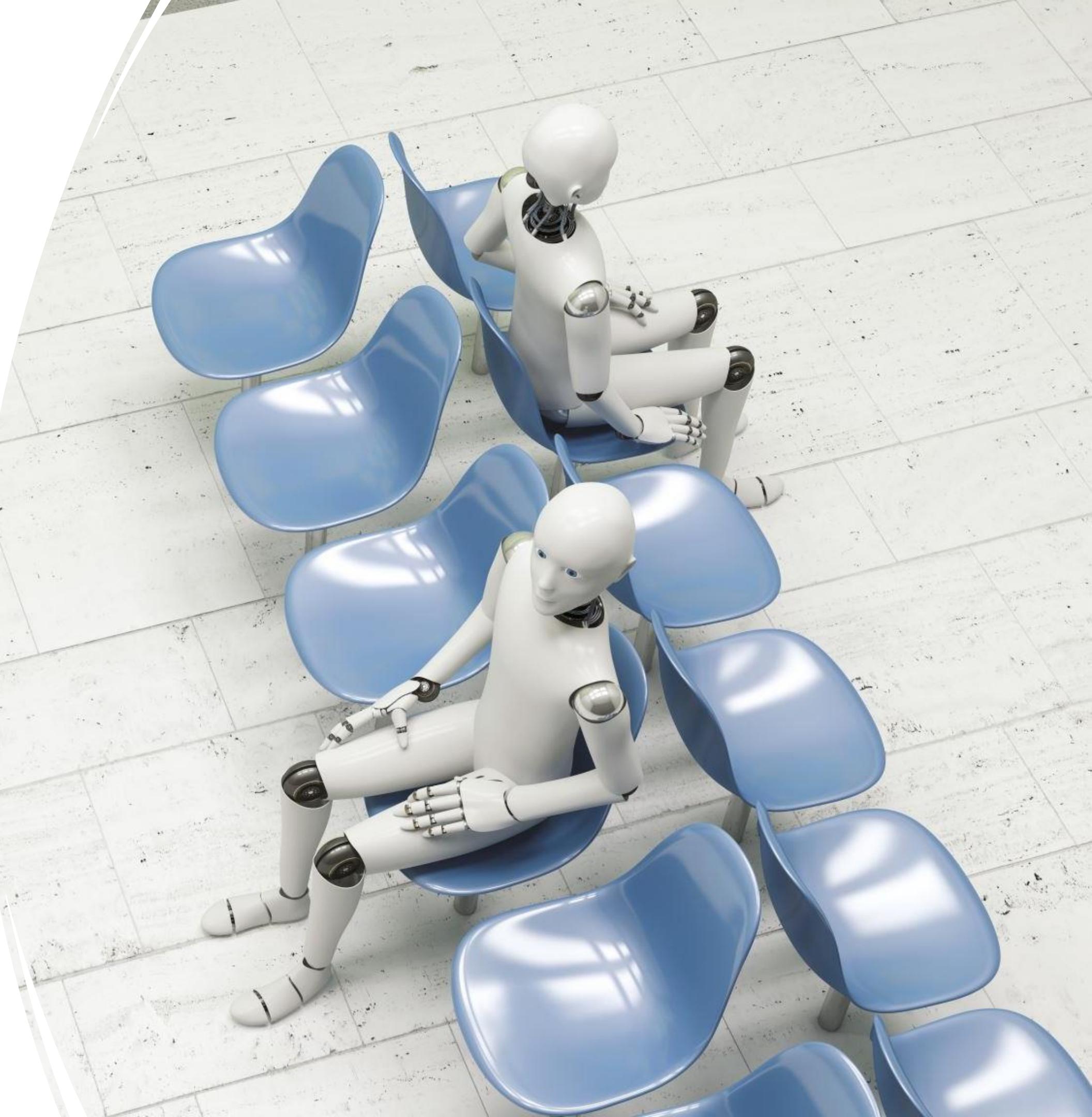
## Example of mission requirement

“After closure, the robots shall clean the electronics store. After cleaning, they shall visit a set of predefined store locations, each at least once, to record the items present on shelves after closure. The robots must minimize the time required to perform this activity. The robots should also patrol the store for security purposes, following any intruder while raising an alarm. The robots should interleave cleaning and security patrolling so that intruders do not remain undetected while the robots are cleaning continually for long periods of time. The robots should monitor their battery, optimize its usage, and recharge when needed. They should avoid recharging simultaneously and leaving the store unmonitored.”

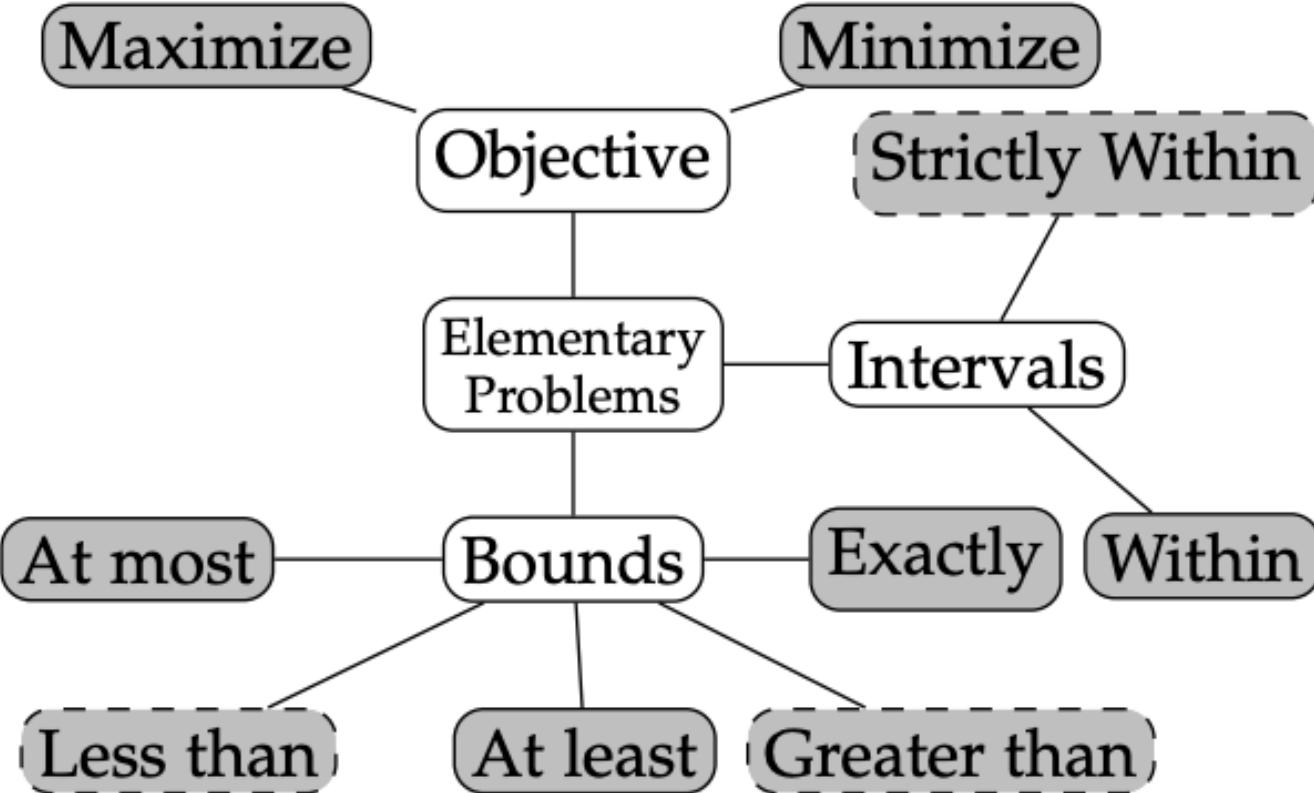
# Support for quantitative aspects

---

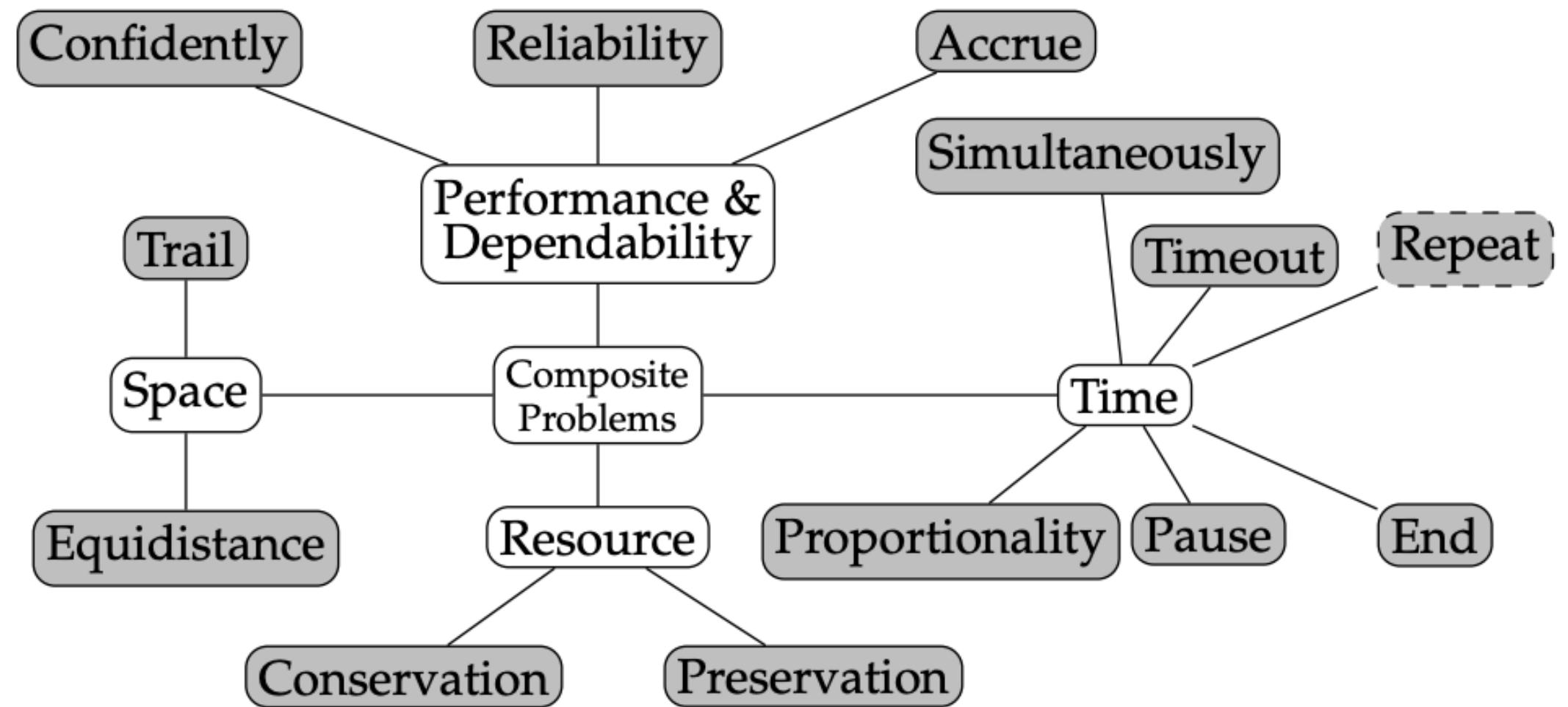
Users and operators of robotic systems often require behaviors that ensure quantitative constraints such as **upper bounds** on the **time** a robot takes to perform an action, the **energy consumption** to complete that action, or the **probability of failing** to achieve a mission goal.



# We extended previous patterns to support the specification of quantitative properties



(a) Elementary mission specification problems.



(b) Composite mission specification problems.

# Domain Specific Language (DSL) including previous patterns

Previous patterns

<b>Mission Pattern</b>	miss ::= miss <b>and</b> miss   miss <b>or</b> miss   <b>not</b> miss   rob <b>shall</b> pat   e_qpat   c_qpat
<b>Elementary Patterns</b>	pat ::= <b>visit</b> ( <b>in sequence</b>   <b>in order</b>   <b>in strict order</b>   <b>fairly</b> )? locs   <b>patrol</b> ( <b>in sequence</b>   <b>in order</b>   <b>in strict order</b>   <b>fairly</b> )? locs   <b>visit</b> ( <b>more than</b>   <b>less than</b>   <b>exactly</b> ) n <b>times</b> loc   <b>avoid</b> (loc <b>until</b> cond   loc   loc <b>after</b> cond)   <b>react</b> ( <b>instantly</b>   <b>with a delay</b>   <b>promptly</b> ) <b>to</b> cond <b>by</b> (exec act   pat   <b>reach</b> loc)   <b>counteract</b> ( <b>instantly</b>   <b>with a delay</b> ) <b>when reach</b> loc <b>by</b> cond <b>wait in location</b> loc <b>until</b> cond
<b>Composite Patterns</b>	e_qpat ::= <b>maximize</b> m miss   <b>minimize</b> m miss   m <b>at most</b> v miss   m <b>less than</b> v miss   m <b>at least</b> v miss   m <b>greater than</b> v miss   m <b>exactly</b> v miss   m <b>within</b> v <sub>1</sub> <b>and</b> v <sub>2</sub> miss   m <b>strictly within</b> v <sub>1</sub> <b>and</b> v <sub>2</sub> miss
<b>Condition Locations</b>	c_qpat ::= <b>conserve</b> m <b>while</b> miss   <b>preserve</b> m <b>within</b> [v <sub>1</sub> ,v <sub>2</sub> ] <b>while</b> miss   <b>pause</b> v miss   <b>timeout</b> v miss   <b>repeat</b> miss <b>every</b> v   <b>end</b> miss <b>exactly at</b> v   <b>time of</b> miss <sub>1</sub> <b>proportional to</b> miss <sub>2</sub> <b>by factor</b> v   <b>execute</b> rob <b>actions</b> act <sub>1</sub> ,act <sub>2</sub> ,...act <sub>n</sub>   rob <b>accrue</b> m <b>while</b> miss   <b>achieve</b> miss <b>with reliability</b> m ( <b>greater</b>   <b>less</b> ) <b>than</b> v   <b>achieve</b> miss <b>with confidence</b> m ( <b>greater</b>   <b>less</b> ) <b>than</b> v   rob miss <b>equidistance</b> rob <sub>1</sub> rob <sub>2</sub>   rob <b>trail o with distance</b> v
	cond ::= condition <b>is true</b>   act <b>is ended</b>   rob <b>in</b> loc
	locs ::= {loc (,loc)*}

\* miss, miss<sub>1</sub>, miss<sub>2</sub> are missions; v, v<sub>1</sub>, v<sub>2</sub> are values; rob is a robot, o is an object, m is the name of the quantitative measure.

# New “quantitative” patterns

Problem	Description	DSL
<i>Maximize</i>	Maximize $m$ while performing the mission $\text{miss}$ .	<b>maximize</b> $m$ $\text{miss}$
<i>Minimize</i>	Minimize $m$ while performing the mission $\text{miss}$ .	<b>minimize</b> $m$ $\text{miss}$
<i>At most</i>	Keep $m$ lower than or equal to $v$ while performing $\text{miss}$ .	$m$ <b>at most</b> $v$ $\text{miss}$
<i>Less than</i>	Keep $m$ strictly lower than $v$ while performing $\text{miss}$ .	$m$ <b>less than</b> $v$ $\text{miss}$
<i>At least</i>	Keep $m$ greater than or equal to $v$ while performing $\text{miss}$ .	$m$ <b>at least</b> $v$ $\text{miss}$
<i>Greater than</i>	Keep $m$ strictly greater than $v$ while performing $\text{miss}$ .	$m$ <b>greater than</b> $v$ $\text{miss}$
<i>Exactly</i>	Keep $m$ exactly $v$ while performing $\text{miss}$ .	$m$ <b>exactly</b> $v$ $\text{miss}$
<i>Within</i>	Keep $m$ within the (closed) interval $[v_1, v_2]$ while performing $\text{miss}$ .	$m$ <b>within</b> $v_1$ <b>and</b> $v_2$ $\text{miss}$
<i>Strictly Within</i>	Keep $m$ within the (open) interval $(v_1, v_2)$ while performing $\text{miss}$ .	$m$ <b>strictly within</b> $v_1$ <b>and</b> $v_2$ $\text{miss}$
<i>Conservation</i>	Minimize the value of $m$ performing $\text{miss}$ .	<b>conserve</b> $m$ <b>while</b> $\text{miss}$
<i>Preservation</i>	Keep the value of $m$ within interval $[b_l, b_u]$ while performing $\text{miss}$ .	<b>preserve</b> $m$ <b>within</b> $[b_l, b_u]$ <b>while</b> $\text{miss}$
<i>Pause</i>	Pause the mission $\text{miss}$ for $v$ time instants. Then, resume it.	<b>pause</b> $v$ $\text{miss}$
<i>Timeout-deadline</i>	Execute $\text{miss}$ . Stop the execution when the timeout $v$ is reached.	<b>timeout</b> $v$ $\text{miss}$
<i>Repeat</i>	Repeat the mission $\text{miss}$ every $v$ time units.	<b>repeat</b> $\text{miss}$ <b>every</b> $v$
<i>End</i>	Terminate mission $\text{miss}$ exactly at time $v$ .	<b>end</b> $\text{miss}$ <b>exactly_at</b> $v$
<i>Proportionality</i>	Keep the time to perform $\text{miss}_1$ and $\text{miss}_2$ proportional by a factor $v$ .	<b>time_of</b> $\text{miss}_1$ <b>proportional_to</b> $v$ $\text{miss}_2$
<i>Simultaneously</i>	Execute the actions $\text{act}_1, \text{act}_2, \dots, \text{act}_n$ simultaneously.	<b>execute</b> $\text{rob}$ <b>actions</b> $\text{act}_1, \text{act}_2, \dots, \text{act}_n$
<i>Accrue</i>	Maximize the performance $m$ while performing $\text{miss}$ .	<b>rob accrue</b> $m$ <b>while</b> $\text{miss}$
<i>Reliably</i>	Ensure that the measure $m$ is higher/lower than the value $v$ .	<b>achieve</b> $\text{miss}$ <b>with reliability</b> $m$ $v$
<i>Confidently</i>	Achieve $\text{miss}$ and ensure that confidence $m$ is higher/lower than $v$ .	<b>achieve</b> $\text{miss}$ <b>with confidence</b> $m$ $v$
<i>Equidistance</i>	$\text{rob}$ performs $\text{miss}$ by keeping $\text{rob}_1$ and $\text{rob}_2$ at the same distance.	$\text{rob miss}$ <b>equidistance</b> $\text{rob}_1$ $\text{rob}_2$
<i>Trail</i>	$\text{rob}$ follows object $\circ$ keeping a distance $v$ .	$\text{rob trail}$ $\circ$ <b>with distance</b> $v$

\*  $\text{miss}, \text{miss}_1, \text{miss}_2$  are missions;  $v, v_1, v_2$  are values;  $\text{rob}$  is a robot,  $\circ$  is an object,  $m$  is the name of the quantitative measure.  
 [...] represents portions of the DSL of Figure 4 omitted for graphical reasons.

# Translation to Probabilistic Reward Computation Tree Logic (PRCTL)

<b>Mission</b>	$\tau(\text{miss1} \text{ and } \text{miss2}) = \tau(\text{miss1}) \wedge \tau(\text{miss2})$	$\tau(\text{miss1} \text{ or } \text{miss2}) = \tau(\text{miss1}) \vee \tau(\text{miss2})$
	$\tau(\text{not miss}) = \neg \tau(\text{miss})$	$\text{rob shall pat} = \tau(\text{pat}[r \leftarrow \text{rob}])$
<b>Elementary Patterns</b>	$\tau(\text{maximize m miss}) = \mathcal{P}_{\max=?}(\tau(\text{miss}))$	$\tau(\text{minimize m miss}) = \mathcal{P}_{\min=?}(\tau(\text{miss}))$
	$\tau(\text{m at most v miss}) = \mathcal{P}_{\leq_v}(\tau(\text{miss}))$	$\tau(\text{m less than v miss}) = \mathcal{P}_{<_v}(\tau(\text{miss}))$
	$\tau(\text{m at least v miss}) = \mathcal{P}_{\geq_v}(\tau(\text{miss}))$	$\tau(\text{m greater than v miss}) = \mathcal{P}_{>_v}(\tau(\text{miss}))$
	$\tau(\text{m exactly v miss}) = \mathcal{P}_{\geq_v}(\tau(\text{miss})) \wedge \mathcal{P}_{\leq_v}(\tau(\text{miss}))$	
	$\tau(\text{m within v}_1 \text{ and } \text{v}_2 \text{ miss}) = \mathcal{P}_{\geq_{v_1}}(\tau(\text{miss})) \wedge \mathcal{P}_{\leq_{v_2}}(\tau(\text{miss}))$	
	$\tau(\text{m strictly within v}_1 \text{ and } \text{v}_2 \text{ miss}) = \mathcal{P}_{>_{v_1}}(\tau(\text{miss})) \wedge \mathcal{P}_{<_{v_2}}(\tau(\text{miss}))$	
<b>Prob.</b>	$\tau(\text{maximize m miss}) = \mathcal{E}_{\max=?}(\tau(\text{miss}))$	$\tau(\text{minimize m miss}) = \mathcal{E}_{\min=?}(\tau(\text{miss}))$
	$\tau(\text{m at most v miss}) = \mathcal{E}_{[0,v]}(\tau(\text{miss}))$	$\tau(\text{m less than v miss}) = \mathcal{E}_{[0,v)}(\tau(\text{miss}))$
	$\tau(\text{m at least v miss}) = \mathcal{E}_{[v,\infty)}(\tau(\text{miss}))$	$\tau(\text{m greater than v miss}) = \mathcal{E}_{(v,\infty)}(\tau(\text{miss}))$
	$\tau(\text{m exactly v miss}) = \mathcal{E}_{\geq_v}(\tau(\text{miss})) \wedge \mathcal{E}_{\leq_v}(\tau(\text{miss}))$	
	$\tau(\text{m within v}_1 \text{ and } \text{v}_2 \text{ miss}) = \mathcal{E}_{[v_1,\infty)}(\tau(\text{miss})) \wedge \mathcal{E}_{[0,v_2]}(\tau(\text{miss}))$	
	$\tau(\text{m strictly within v}_1 \text{ and } \text{v}_2 \text{ miss}) = \mathcal{E}_{(v_1,\infty)}(\tau(\text{miss})) \wedge \mathcal{E}_{[0,v_2)}(\tau(\text{miss}))$	
<b>Rewards</b>	$\tau(\text{conserve m while miss}) = \mathcal{E}_{\min=?}(\tau(\text{miss}))$	
	$\tau(\text{preserve m within [v}_1, \text{v}_2] \text{ while miss}) = \mathcal{E}_{[v_1,v_2]}(\tau(\text{miss}))$	
	$\tau(\text{pause v miss}) = \mathcal{G}^{[0,v]} \tau(\neg \text{miss}) \wedge (\mathcal{F}^{[v+1,v+1]}(\tau(\text{miss})))$	
	$\tau(\text{timeout v miss}) = \mathcal{G}^{[v,\infty)}(\neg \tau(\text{miss}))$	
	$\tau(\text{repeat miss every v}) = \tau(\text{miss}) \wedge \mathcal{G}^{[0,\infty]}(\tau(\text{miss}) \rightarrow (\mathcal{G}^{[1,v-1]}(\neg \tau(\text{miss})) \wedge (\mathcal{F}^{[v,v]}(\tau(\text{miss}))))))$	
	$\tau(\text{end miss exactly at v}) = \mathcal{G}^{[0,v)}(\tau(\text{miss})) \wedge \mathcal{G}^{[v,\infty]}(\neg \tau(\text{miss}))$	
<b>Composite Patterns</b>	$\tau(\text{time of miss}_1 \text{ proportional to miss}_2 \text{ by factor v}) = \text{NA}$ (Not Available in PRCTL)	
	$\tau(\text{execute rob actions act}_1, \text{act}_2, \dots, \text{act}_n) = \mathcal{F}(\bigwedge_{i=1}^n \text{act}_i)$	
	$\tau(r \text{ accrue m while miss}) = \mathcal{E}_{\max=?}(\tau(\text{miss}))$	
	$\tau(\text{achieve miss with reliability m (greater   less) than v}) = \mathcal{E}_{[v,\infty)}(\tau(\text{miss}))/\mathcal{E}_{[0,v)}(\tau(\text{miss}))$	
	$\tau(\text{achieve miss with confidence m (greater   less) than v}) = \mathcal{L}_{>v}(\tau(\text{miss}))/\mathcal{L}_{<v}(\tau(\text{miss}))$	
	$\tau(\text{rob miss equidistance rob}_1 \text{ rob}_2) = \text{NA}$ (Not Available in PRCTL)	
	$\tau(\text{rob trail o with distance v}) = \text{NA}$ (Not Available in PRCTL)	

# Further info about specification patterns

A line drawing of four musicians playing string instruments: a violinist, a violist, a cellist, and a double bass player. They are arranged in a row, facing right, as if performing in a quartet.

This page complements the manuscript "Robotic Mission Specification Patterns: Providing Support for Quantitative Properties" and is an online repository of a quantitative specification pattern catalog for missions of mobile robots, along with an accompanying DSL and tool support: QUARTET. The pattern system is not intended to be exhaustive or complete, and the repository is not intended to be static. The set of patterns will grow over time as designers specify missions that do not belong to the provided patterns.

You can further find the patterns, information on requirements collection as well as DSL and tool support through QUARTET. Reproduction kits, specifications and accompanying code can be found in evaluation. See also an introductory video to QUARTET.

<https://roboticpatterns.com/quantitative>



<https://github.com/Gricel-lee/Quartet-MRS-DSL>



Claudio Menghi, Christos Tsigkanos, Mehrnoosh Askarpour , Patrizio Pelliccione, Gricel Vazquez , Radu Calinescu, and Sergio García "Mission Specification Patterns for Mobile Robots: Providing Support for Quantitative Properties," in **IEEE Transactions on Software Engineering (TSE)**, doi: 10.1109/TSE.2022.3230059.

# Mission Specification Patterns for Mobile Robots: Providing Support for Quantitative Properties

di Menghi, Christos Tsigkanos, Mehrnoosh Askarpour, Patrizio Pelliccione, Gricel Vazquez, Radu Calinescu, and Sergio García

**Abstract**—With many applications across domains as diverse as aerospace, automotive, and robotics, mission planning has become a critical demand. Nevertheless, the design of these robots often struggle with the lack of a formalized language to express their requirements and sufficiently precise to enable automatic verification and planning of their behaviors. In this paper, we propose QUARTET, a mission-oriented framework of robotics missions that use the mission specification language (MSL) to define the mission requirements. The MSL is a formal language that uses a set of primitives to define the mission requirements. For instance, the *planning of a perimeter*, the *availability of a resource*, or the *reaching of a target*. The QUARTET framework introduces a catalog of *Observation*, *Robotics*, *mission specification*, and *Planning* blocks. The *Observation* block is used to define the challenge of specifying the reliability, performance, resource usage, and cost of a mission. The *Robotics* block is used to define the robot's behavior and its interaction with the environment. The *mission specification* block is used to define the mission requirements. The *Planning* block is used to define the planning of the mission. The QUARTET framework is evaluated in two scenarios: the *reinforcement learning of a robot's behavior* and the *robotic mission planning*. The results show that the QUARTET framework is able to automatically generate mission specifications from raw logic. The QUARTET framework is also able to automatically generate mission specifications from raw logic.

**Index Terms**—Robotics Software Engineering, Robot Mission Specification, Quantitative Properties, Domain-Specific Languages, Planning, Probabilistic Reasoning, Formal Logic.

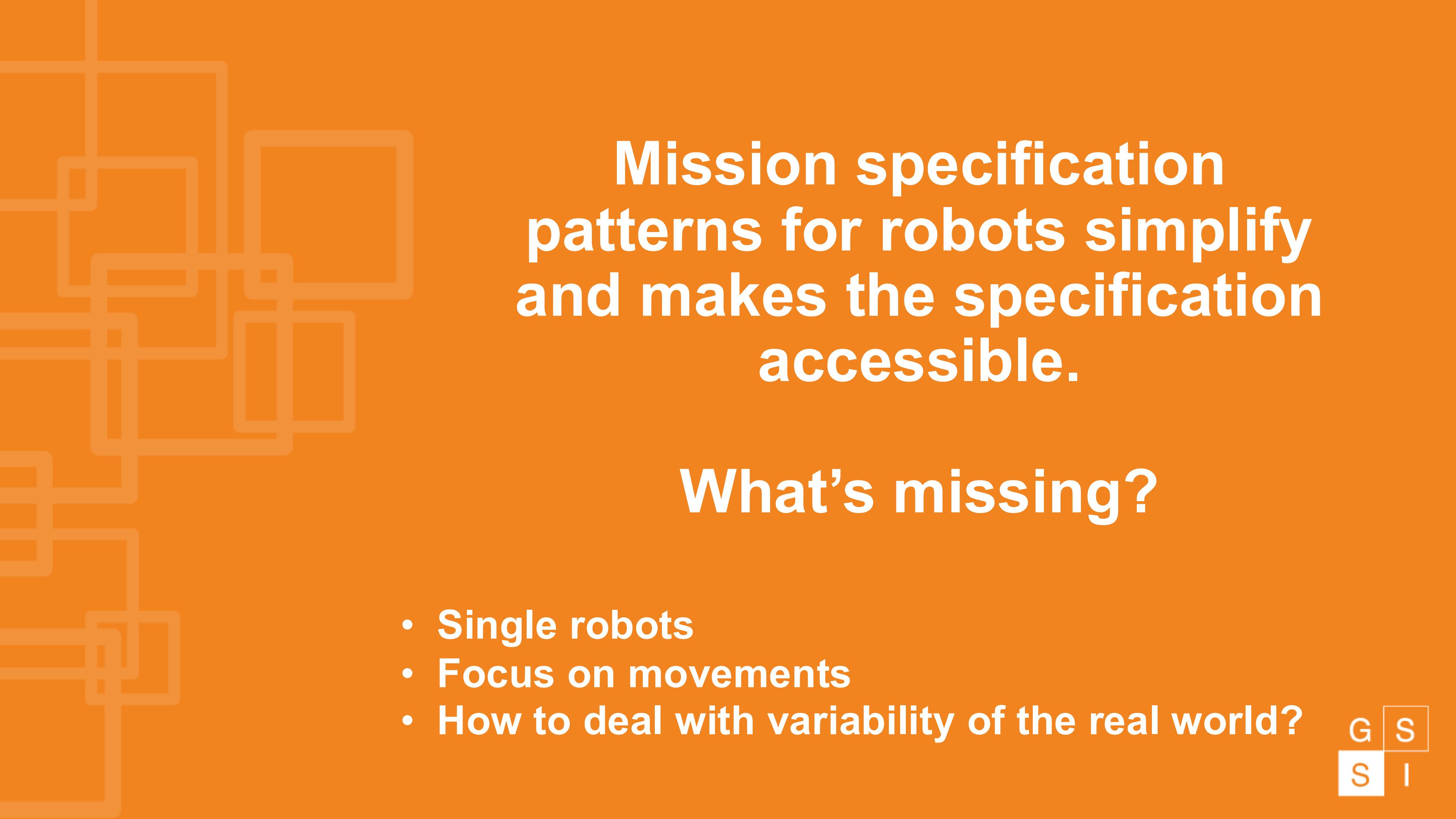
**T**HE engineering of robotic applications is a complex interdisciplinary activity. Similar to other domains, robotics requires continuous communication between interdisciplinary teams. Robotics engineers build low-level primitives that allow higher-order control, while the engineering team developing a robotic application or other stakeholders, (i) verification, where the robotic application is checked against its requirements or system or specifications are checked against the specifications, and (ii) synthesis, where behavior that possibly satisfy the specification is generated.

Mission operations are often experienced in domain-specific languages, such as ROS [1], which was proposed in the last decade [14, 16]. These DLSs are usually integrated with development environments, enabling users to quickly prototype and validate their systems with real robots [19, 20, 21, 21, 22]. However, these languages are typically bound to specific types of robots, and support a limited class of missions. Moreover,



Mission specification  
patterns for robots simplify  
and makes the specification  
accessible.

What's missing?



**Mission specification  
patterns for robots simplify  
and makes the specification  
accessible.**

**What's missing?**

- Single robots
- Focus on movements
- How to deal with variability of the real world?

# Two steps

What kind of missions are specified in practice?

- Identification of missions already specified in practice (i.e. papers, documents of robotic companies)
- Definition of a catalogue of mission specification patterns
- Tool support for assisting users in the specification of missions via the use and instantiation of patterns

How to use these patterns to specify complex missions?

- Definition of operators to combine the mission specification patterns
- Definition of a Domain Specific Language (DSL) with graphical and textual syntax
- Definition of a tool support for the DSL

# Two steps

patterns

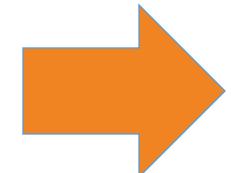
What kind of missions are specified in practice?

- Identification of missions already specified in practice (i.e. papers, documents of robotic companies)
- Definition of a catalogue of mission specification patterns
- Tool support for assisting users in the specification of missions via the use and instantiation of patterns

How to use these patterns to specify complex missions?

- Definition of operators to combine the mission specification patterns
- Definition of a Domain Specific Language (DSL) with graphical and textual syntax
- Definition of a tool support for the DSL

# Two steps



What kind of missions are specified in practice?

- Identification of missions already specified in practice (i.e. papers, documents of robotic companies)
- Definition of a catalogue of mission specification patterns
- Tool support for assisting users in the specification of missions via the use and instantiation of patterns

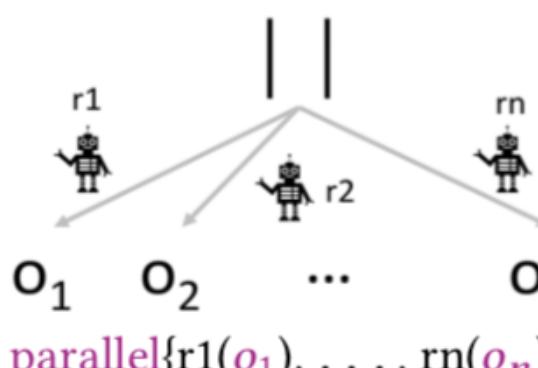
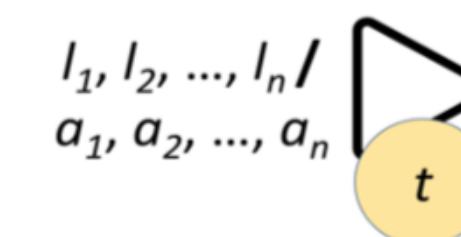
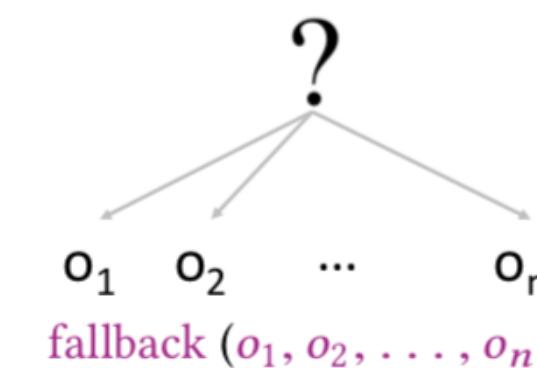
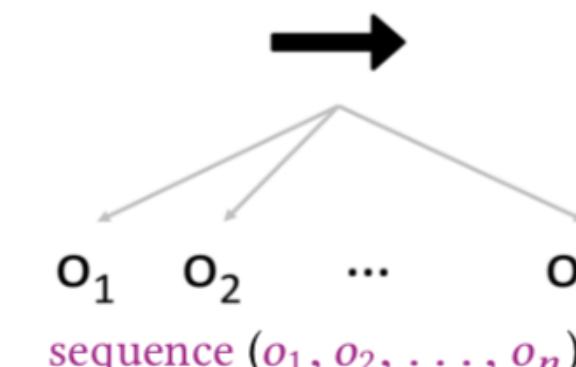
How to use these patterns to specify complex missions?

- Definition of operators to combine the mission specification patterns
- Definition of a Domain Specific Language (DSL) with graphical and textual syntax
- Definition of a tool support for the DSL

# Domain Specific Language to specify missions

- PROMISE (simPle RObot MIssion SpEcification)
  - Patterns are basic building blocks
  - Operators enable the composition of patterns towards the specification of complex missions for multi-robots

# Operators of the DSL

Name	Description	Semantics	Syntax	Intermediate language
Parallel $\parallel_{(r_1, \dots, r_n, o_1, \dots, o_n)}$	Always the root of the mission. The operators $o_1, o_2, \dots, o_n$ are executed in parallel, each by a different robot—i.e., assigns one branch to each robot. Returns success when all operators return success, failure otherwise.	$\{res_1, res_2, \dots, res_n\} = \{o_1, o_2, \dots, o_n\}$ <b>if</b> ( $res_1 == \top \wedge \dots \wedge res_n == \top$ ) <b>then</b> <b>return</b> $\top$ <b>else return</b> $\perp$	 <p><math>o_1 \quad o_2 \quad \dots \quad o_n</math>  <b>parallel</b> {<math>r_1(o_1), \dots, r_n(o_n)</math>}</p>	$r_1[o_1]$ $r_2[o_2]$ $\dots$ $r_n[o_n]$
Delegate $\triangleright(\mathcal{E}, t)$	Delegates execution of a task $t$ to a specific robot (specified by the Parallel operator). Tasks are specified using patterns for robotic missions that take as input parameters as locations (indicated as $l_1, l_2, \dots, l_n$ ) and actions (indicated as $a_1, a_2, \dots, a_n$ ).	<b>execute</b> ( $\mathcal{E}, t$ )	 <p><math>l_1, l_2, \dots, l_n /</math>  <math>a_1, a_2, \dots, a_n</math></p> <p><b>delegate</b>(<math>t</math> locations <math>l_1, \dots, l_n</math>)  <b>delegate</b>(<math>t</math> actions <math>a_1, \dots, a_n</math>)</p>	LTL formula of the pattern specified by the task $t$ .
Fallback $?(\{o_1, o_2, \dots, o_n\})$	Executes the first operator; if it is executed successfully, ends with success. If the execution of the first operator fails, tries to execute the second operator. This procedure is repeated for all the other operators. Returns failure if all operators fail.	<b>if</b> ( $\{o_1, o_2, \dots, o_n\} \neq \emptyset$ ) <b>then</b> $res = o_1;$ <b>if</b> ( $res == \perp$ ) <b>then</b> $?(\{o_2, \dots, o_n\})$ <b>else return</b> $\top$ <b>else return</b> $\perp$	 <p><math>o_1 \quad o_2 \quad \dots \quad o_n</math>  <b>fallback</b> (<math>o_1, o_2, \dots, o_n</math>)</p>	$parent[fb]$ $fb\_1[o_1]$ $fb\_2[o_2]$ $\dots$ $fb\_n[o_n]$
Sequence $\uparrow (\{o_1, o_2, \dots, o_n\})$	Executes all the operators from the first to the last. If an operator returns success executes the subsequent operator. If an operator returns a failure returns failure. Returns success if and only if all the operators return success.	<b>if</b> ( $\{o_1, o_2, \dots, o_n\} \neq \emptyset$ ) <b>then</b> $res = o_1;$ <b>if</b> ( $res == \top$ ) <b>then</b> $\rightarrow (\{o_2, \dots, o_n\})$ <b>else return</b> $\perp$ <b>else return</b> $\perp$	 <p><math>o_1 \quad o_2 \quad \dots \quad o_n</math>  <b>sequence</b> (<math>o_1, o_2, \dots, o_n</math>)</p>	[ $o_1, o_2, \dots, o_n$ ]

# Operators of the DSL

<p><b>EventHandler</b>  <math>\uparrow(e_1, \dots, e_n, o, o_1, \dots, o_n)</math></p>	<p>Executes a by default operator <math>o</math>. Once an event <math>e_i</math> occurs, executes operator <math>o_i</math> in response. Once the execution of <math>o_i</math> is finished, resumes the operator <math>o</math>. Returns success if the operator <math>o</math> succeeds and all the events that occurred during the execution of <math>o</math> are correctly handled.</p>	<pre> res = ⊥; <b>while</b>(res ≠ ⊤)     res = o;     <b>if</b>(res == ⊤) <b>then</b>         <b>return</b> ⊤     <b>if</b>(<math>e_i == \top</math>), <b>then</b> <math>i = 1, \dots, n</math>         resint = <math>o_i</math>;         <b>if</b>(resint == ⊥), <b>then</b>             <b>return</b> ⊥         res = <b>resume</b>(<math>o</math>); <b>return</b> res </pre>	<p>parent[<b>eh</b>]  <b>eh_default</b>[<b>o</b>]  <b>eh_e1</b>[<b>o1</b>]  <b>eh_e2</b>[<b>o2</b>]  ...  <b>eh_en</b>[<b>on</b>]</p>
<p><b>Condition</b>  <math>\oplus(\{e_1, \dots, e_n, o_1, \dots, o_n\})</math></p>	<p>Evaluates the conditions from the first to the last. If the evaluation of one or more conditions is true, executes the corresponding operators. Returns <math>\perp</math> if an operation is not successful, i.e., either it fails or an event occurs. Returns <math>\top</math> when all the executed operations return <math>\top</math>.</p>	<pre> <b>if</b>(<math>e_1 == \top</math>) <b>then</b>     res = <math>o_1</math>     <b>if</b>(res == ⊥) <b>then</b>         <b>return</b> ⊥     ... <b>if</b>(<math>e_n == \top</math>) <b>then</b>     res = <math>o_n</math>     <b>if</b>(res == ⊥) <b>then</b>         <b>return</b> ⊥ <b>return</b> ⊤ </pre>	<p>parent[<b>cond</b>]  <b>cond_e1</b>[<b>o1</b>]  <b>cond_e2</b>[<b>o2</b>]  ...  <b>cond_en</b>[<b>on</b>]</p>
<p><b>TaskComb.</b>  <math>\&amp;(\{o_1, o_2\})</math></p>	<p>Allows the composition of a <i>core movement</i> task with one or more <i>avoidance</i> tasks and with one or more <i>trigger</i> tasks. The composition is performed by means of the <i>and</i> logical operator.</p>	<pre> res = <math>o_1 \&amp;&amp; o_2 \&amp;&amp; \dots o_n</math> <b>if</b>(res == ⊤) <b>then</b>     <b>return</b> ⊤ <b>else return</b> ⊥ </pre>	<p>[<b>o1</b> <b>&amp;&amp;</b> <b>o2</b> <b>&amp;&amp;</b> ...  <b>on</b>]</p>

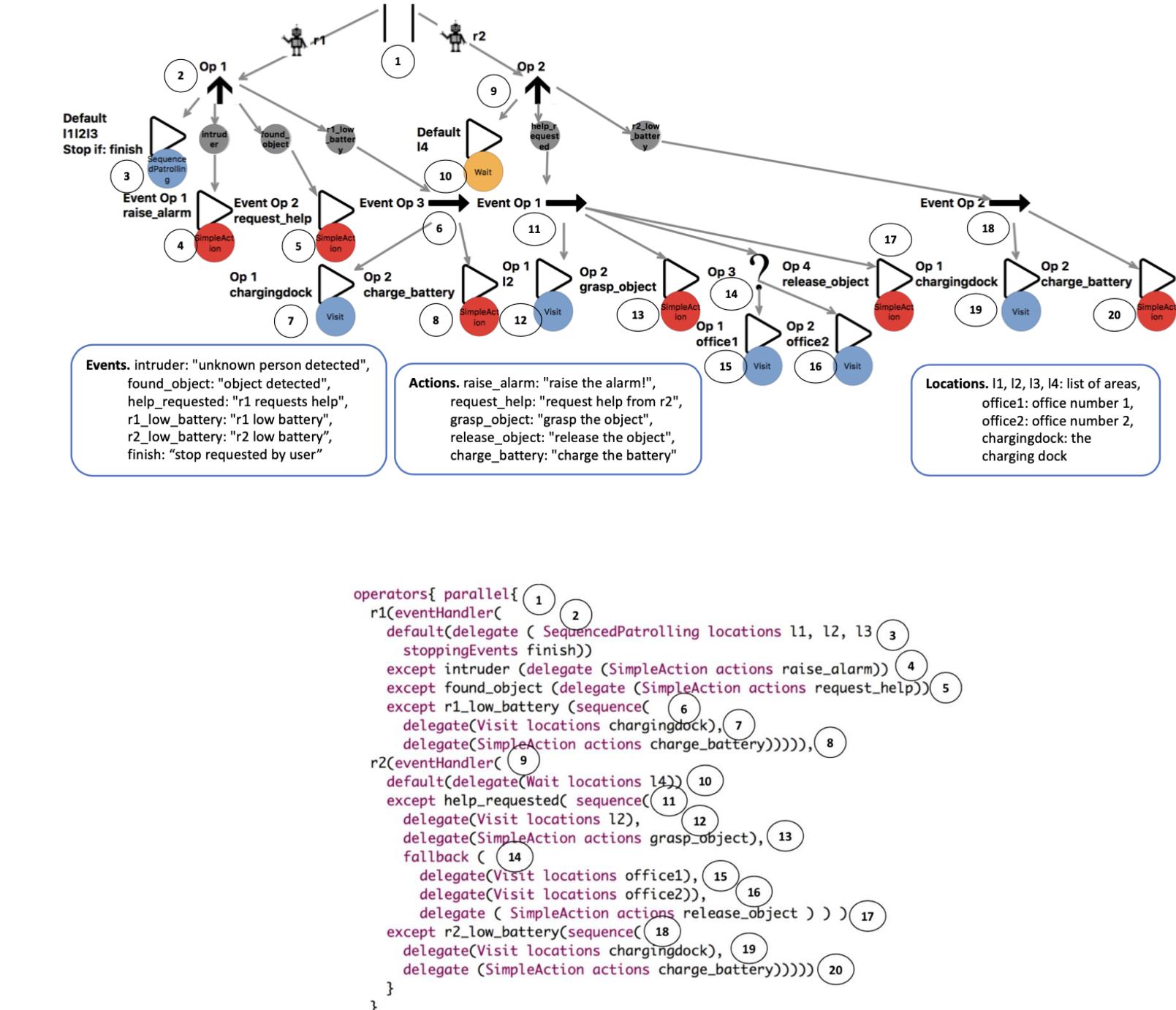
```

Mission:
'mission' '{'
  ('conditions' '{' ('events' events+=Event ( "," events+=Event )*)?
  ('actions' actions+=Action ( "," actions+=Action )*)? '}')?
  'robots' robots+=Robot ( "," robots+=Robot )*
  ('locations' locations+=Location ( "," locations+=Location )*)?
  'operators' '{' operator+=Operator ( "," operator+=Operator )* '}'
'';

Operator:
FallBackOp | SequenceOp | ParallelOp | EventHandlerOp |
ConditionOp | DelegateOp | TaskCombinationOp;
Tasks:
// List of tasks from the provided catalog
Robot:
  name=EString;
Location:
  name=EString;
Event:
  name=ID ':' description=EString;
Action:
  name=ID ':' description=EString;
FallBackOp:
  'fallback' '(' inputOperators+=Operator
  ( "," inputOperators+=Operator )* ')';
SequenceOp:
  'sequence' '(' inputOperators+=Operator
  ( "," inputOperators+=Operator )* ')';
ParallelOp:
  'parallel'
  '{' (inputRobots+=[Robot|EString] '(', inputOperators+=Operator ')
  ( ",," inputRobots+=[Robot|EString] '(', inputOperators+=Operator ')')
  *)? '}';
EventHandlerOp:
  'eventHandler' '('
  'default' '(' inputOperators+=Operator ')'
  ('except' inputEvents+=EventAssignment)+ ')';
ConditionOp:
  'condition' '('
  ('if' inputEvents+=EventAssignment )+ ')';
TaskCombinationOp:
  'combination' '(' inputOperators+=Operator
  (( '&' | 'AND' | 'and' ) inputOperators+=Operator )+ ')';
DelegateOp:
  'delegate' '(' task=Tasks
  ('locations' inputLocations+=[Location|EString]
  ( ",," inputLocations+=[Location|EString] )*)?
  ('actions' inputAction+=[Action|EString]
  ( ",," inputAction+=[Action|EString] )*)?
  ('stoppingEvents' stoppingEvent+=[Event|EString]
  ( ",," stoppingEvent+=[Event|EString] )*)? ')';
EventAssignment:
  inputEvent=[Event|EString] '(' inputOperators=Operator ')';

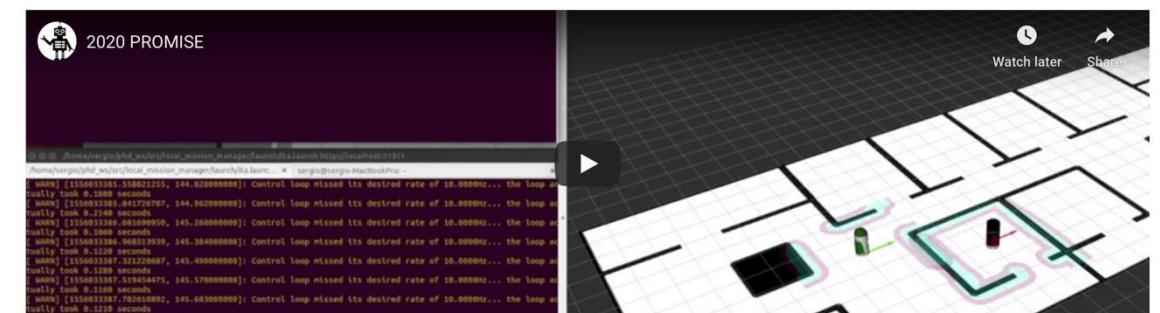
```

## Grammar (Abstract syntax)



## Two concrete syntaxes (Graphical – behaviour tree style - and Textual)

# Further info about Promise



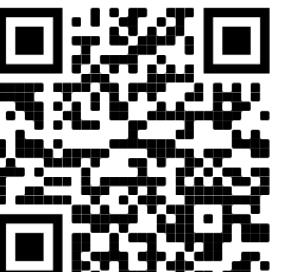
In this page, we present PROMISE (simPle RObot Mission SpEcification), a mission specification language and tool for teams of multiple robots, which is developed as an Eclipse plugin. With our research, we aim at providing a simple yet powerful and rigorous tool to specify, generate, and decompose missions for robotic teams. With this in mind, we integrated PROMISE into a software framework that allows not only mission specification but also execution. This framework is introduced [here](#).

PROMISE was developed to support both developers—i.e., users with programming skills—and non-technical end users—i.e., users who are not necessarily knowledgeable on programming languages—in mission specification.

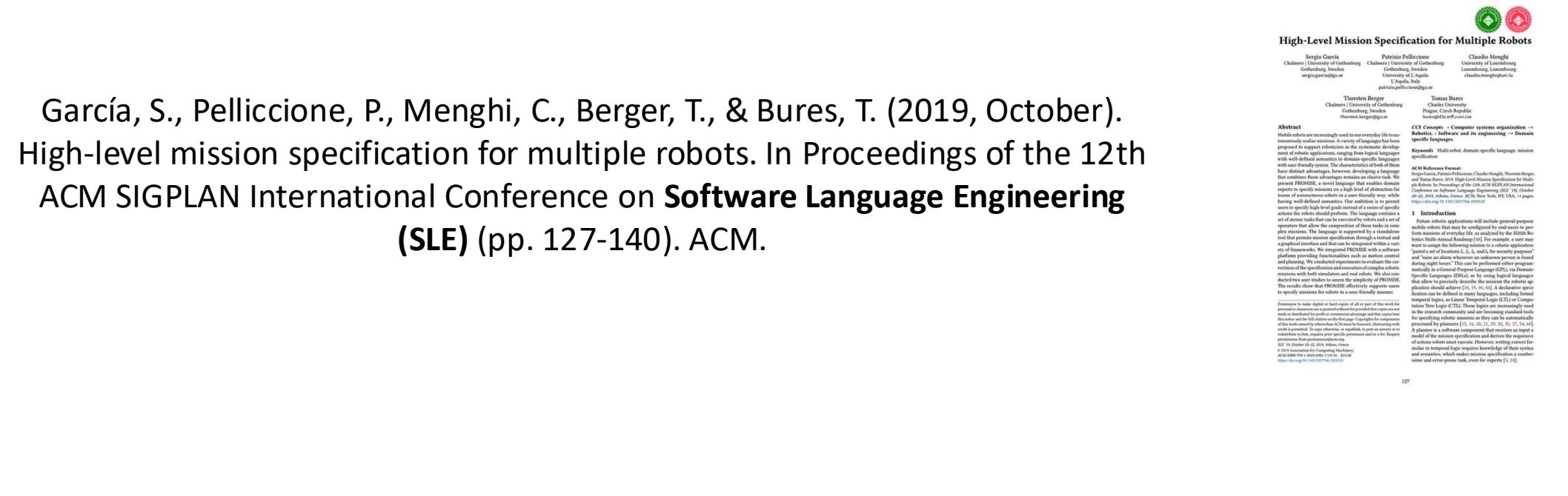
Our DSL supports the specification of complex missions via the use of [a list of operators we proposed](#) that permit the composition of tasks. These operators are inspired by behaviour tree operators [1], which are used in computer science, robotics, control systems and video games for structuring and model behaviors directed toward achieving goals. In turn, the tasks are implemented from [an existing catalog of mission specification patterns](#). To illustrate the mission specification syntaxes of PROMISE we provide a [detailed example](#) in this website.

This page also provides details on the [validation processes](#) we followed during the study and development of PROMISE.

<https://sites.google.com/view/promise-dsl/home>



García, S., Pelliccione, P., Menghi, C., Berger, T., & Bures, T. (2019, October). High-level mission specification for multiple robots. In Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering (SLE) (pp. 127-140). ACM.

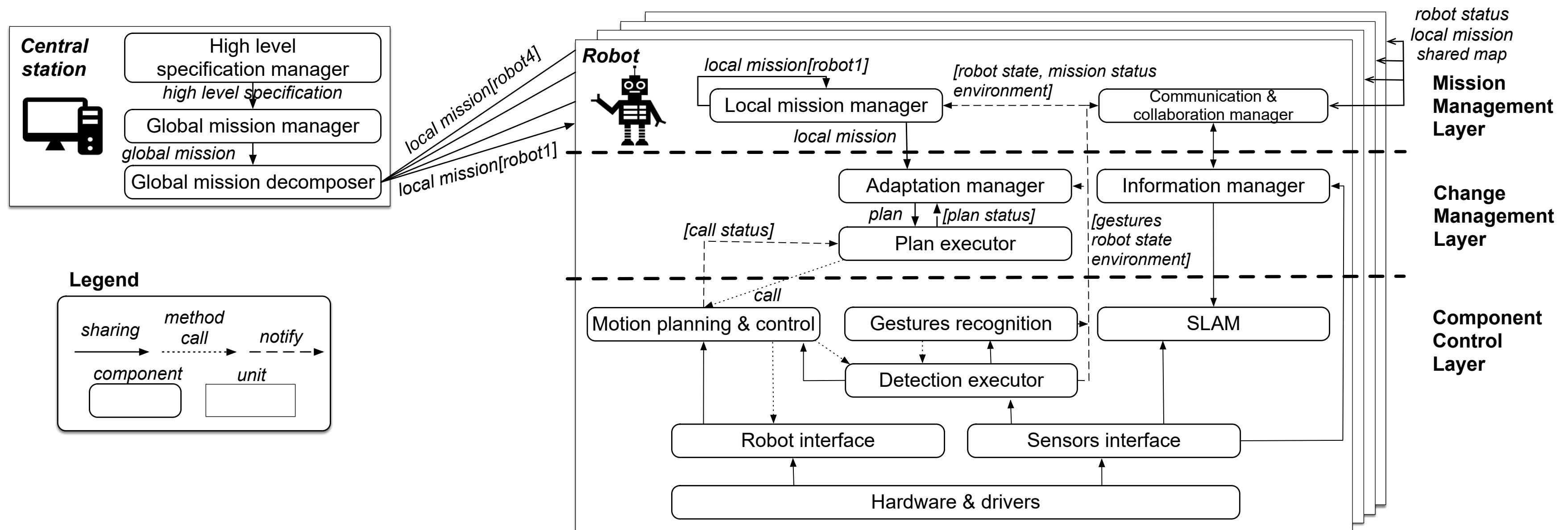


García, S., Pelliccione, P., Menghi, C., Berger, T., & Bures, T. (2020). PROMISE: High-Level Mission Specification for Multiple Robots. In 2nd International Conference on Software Engineering Companion (ICSE '20 Demo).

[https://github.com/SergioGarG/PROMISE\\_implementation](https://github.com/SergioGarG/PROMISE_implementation)



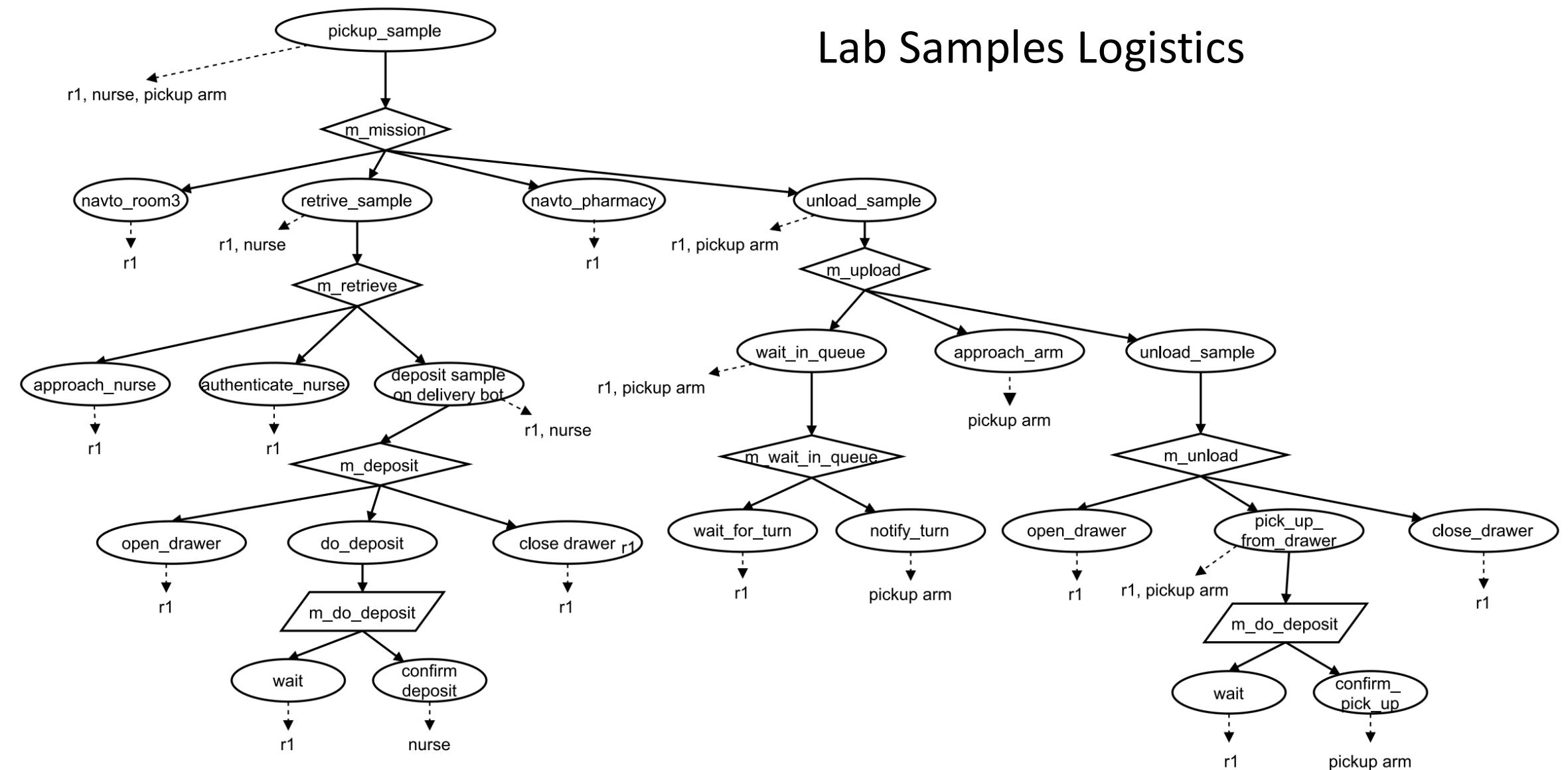
# SERA (Self- adaptive dEcentralized Robotic Architecture)



<https://co4robots.eu/>

# Instantiated Hierarchical Task Networks (iHTN)

- Hierarchical Task Networks is a formalism for task planning.
- The Instantiated HTN (iHTN) formalism formalizes a multi-robot collaborative mission.
- Tasks (ellipses) are efforts that a set of agents (robots or humans) must undertake.
- A task can be abstract or concrete.
- Abstract tasks are refined by methods.
- Methods are linked to tasks of a lower level and a type of ordering.
- The ordering can be sequential (diamond) or unordered (parallelogram).
- ...

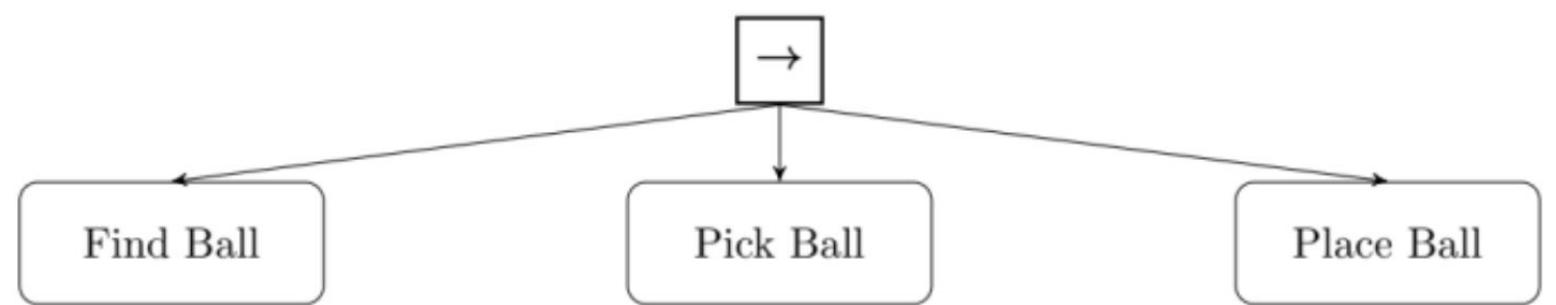




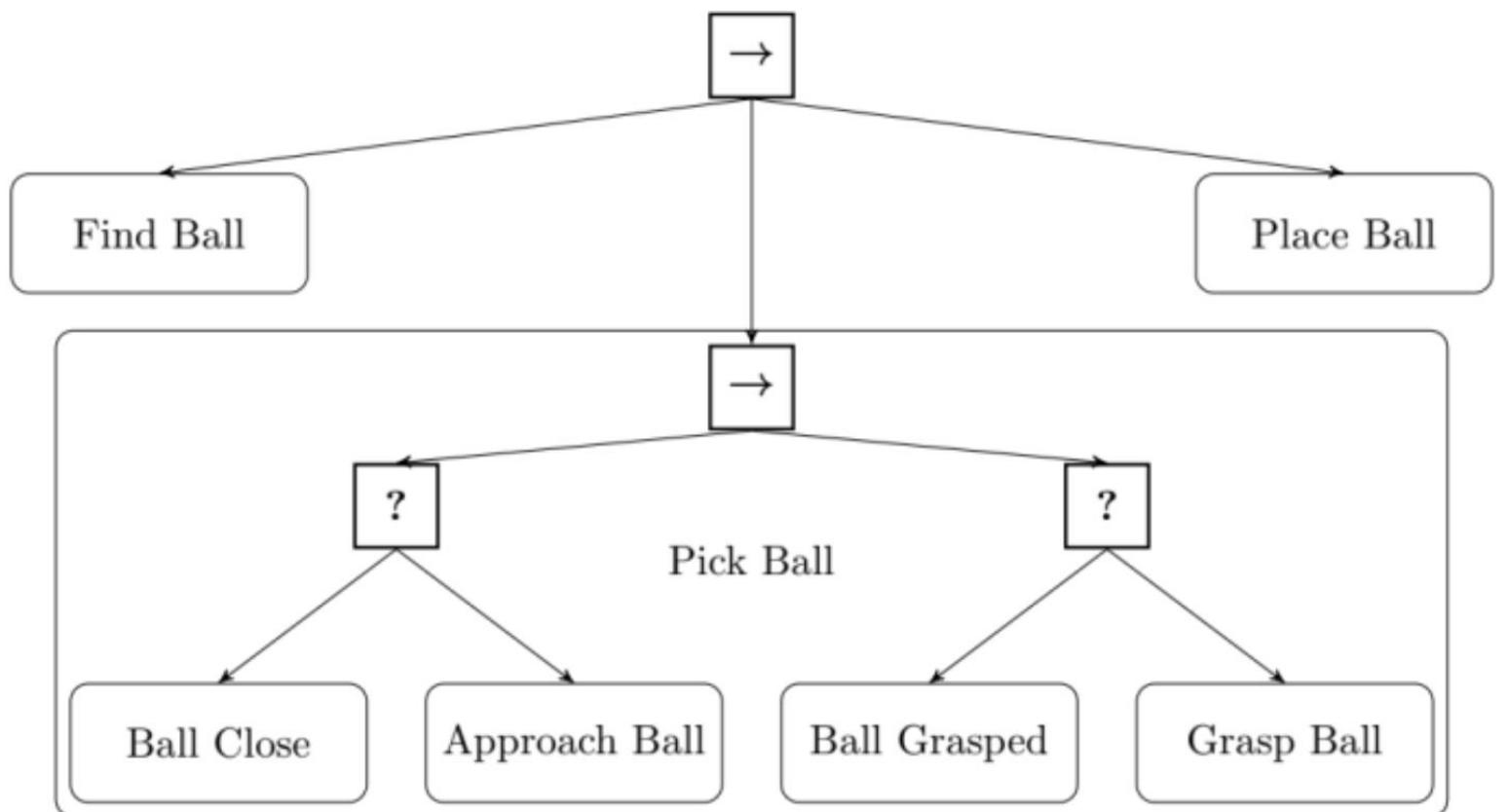
Michele Colledanchise  
Petter Ögren

CRC Press  
Taylor & Francis Group  
A CHAPMAN & HALL BOOK

# Behavior trees



(a) A high level BT carrying out a task consisting of first finding, then picking and finally placing a ball.



(b) The Action Pick Ball from the BT in Figure 1.1(a) is expanded into a sub-BT. The Ball is approached until it is considered close, and then the Action Grasp is executed until the Ball is securely grasped.

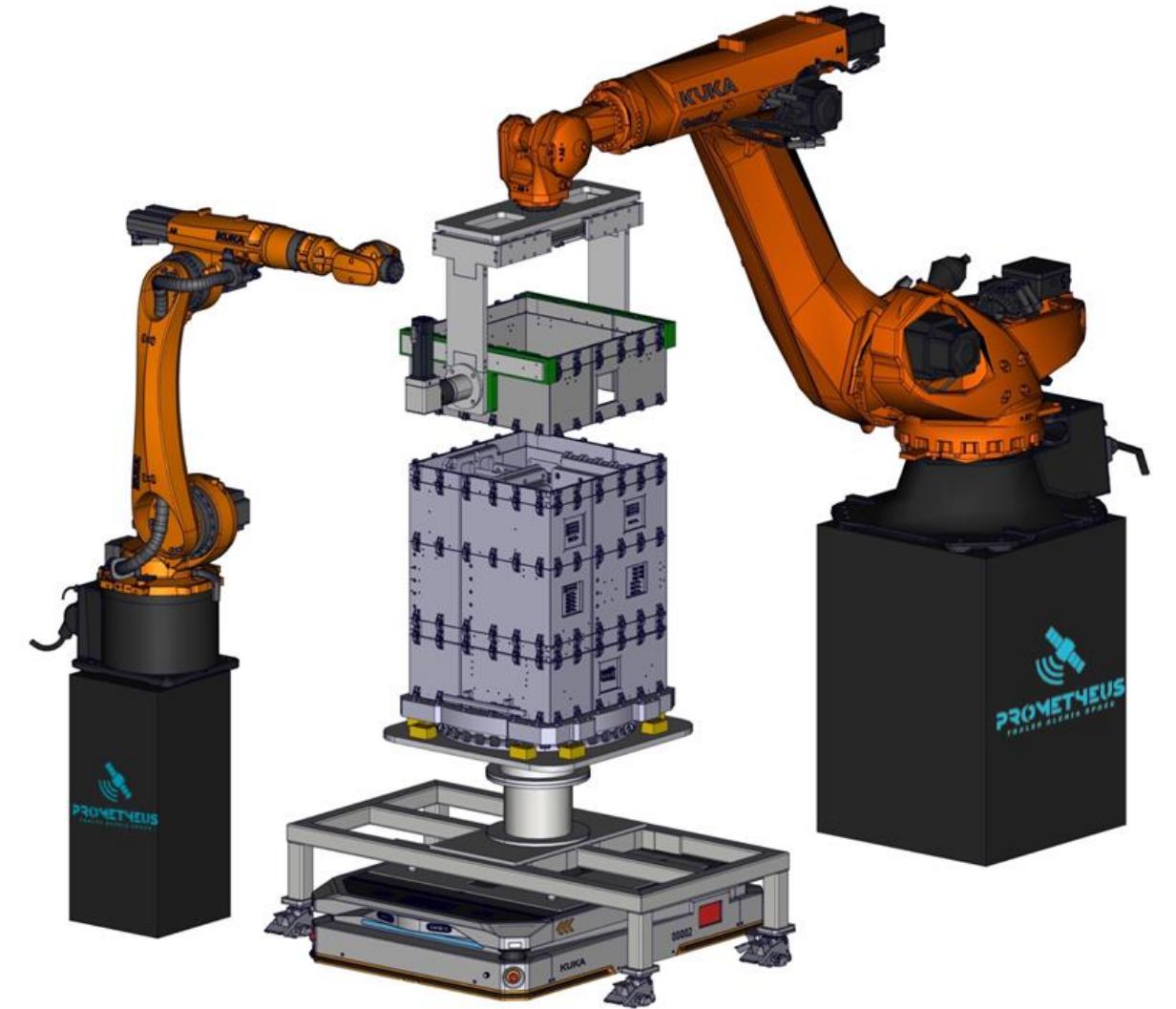
Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	→	If all children succeed	If one child fails	If one child returns Running
Parallel	⇒	If $\geq M$ children succeed	If $> N - M$ children fail	else
Action	text	Upon completion	If impossible to complete	During completion
Condition	text	If true	If false	Never
Decorator	◊	Custom	Custom	Custom



# Democratizing the programming and use of Industrial Robots

# Democratization of Robot Engineering for Advanced Manufacturing (manufacturing satellites)

- Accessible by users without expertise in ICT or robotic
- Coordination of multi and heterogeneous robots and human operators
- It forces modularity and programming with reuse (parametric APIs)



# Domain Specific Language components



Agents (Robots & Operators):

- Robot1, Robot2, HumanOP1, AMR1



Locations:

- Quality Control, Assembly Location, Warehouse, Stacking Platform, Buffer Area



Trays & Components:

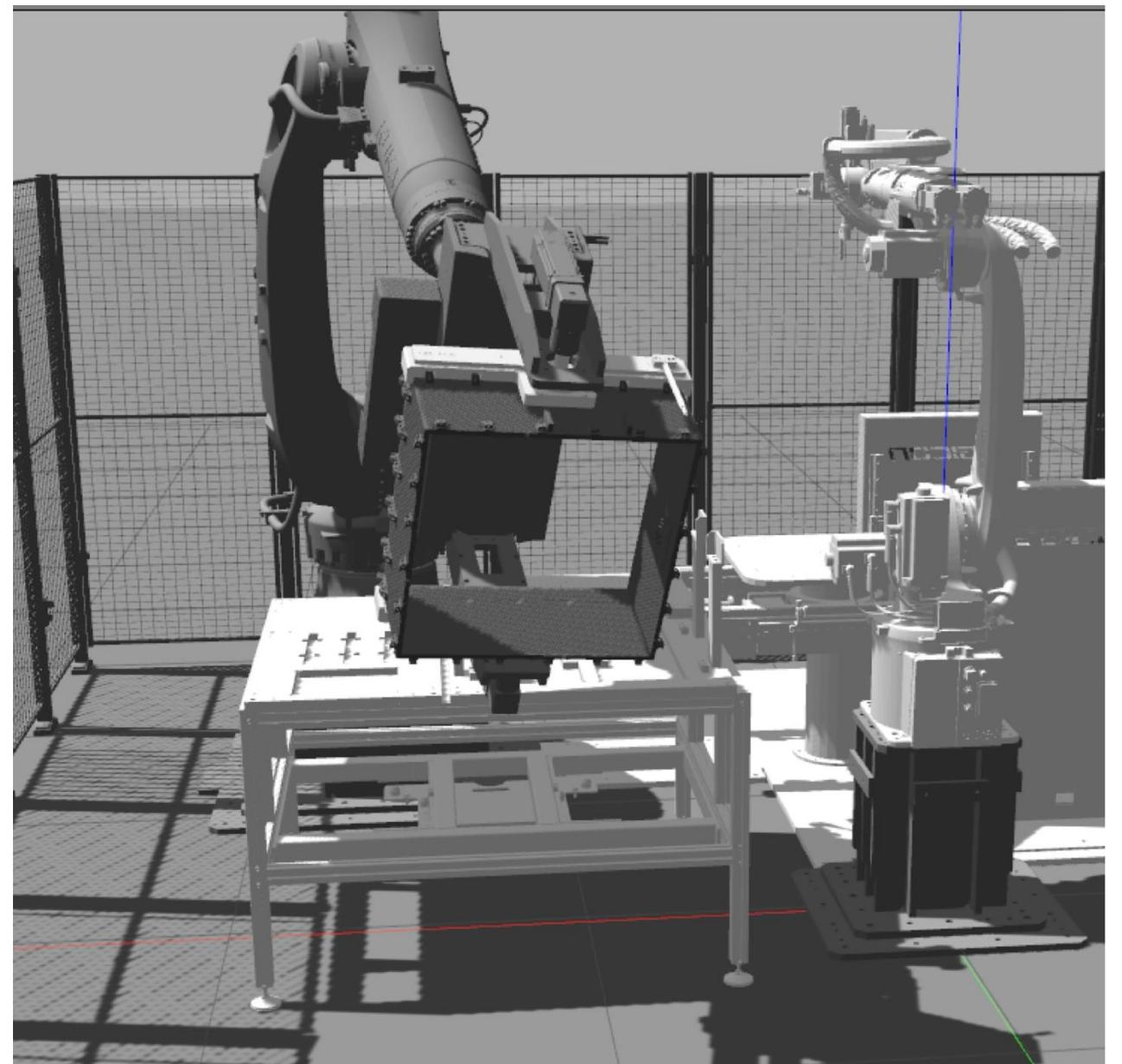
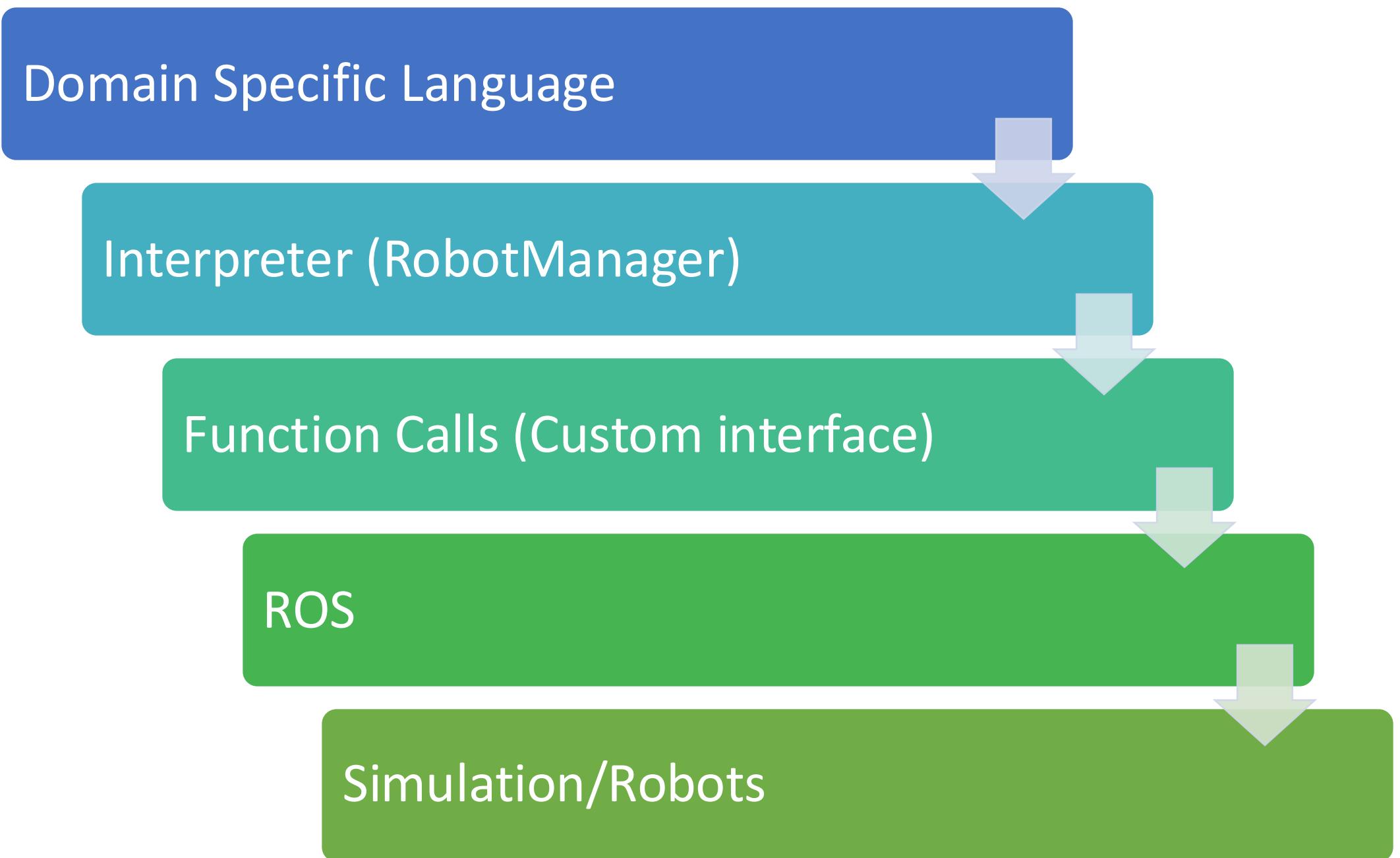
- AOCS Tray, DHC Tray, Components (CMG, MTQ, MAG, screws)



Mission Tasks:

- Moving, Picking, Placing, Screwing, Assembling

# Workflow



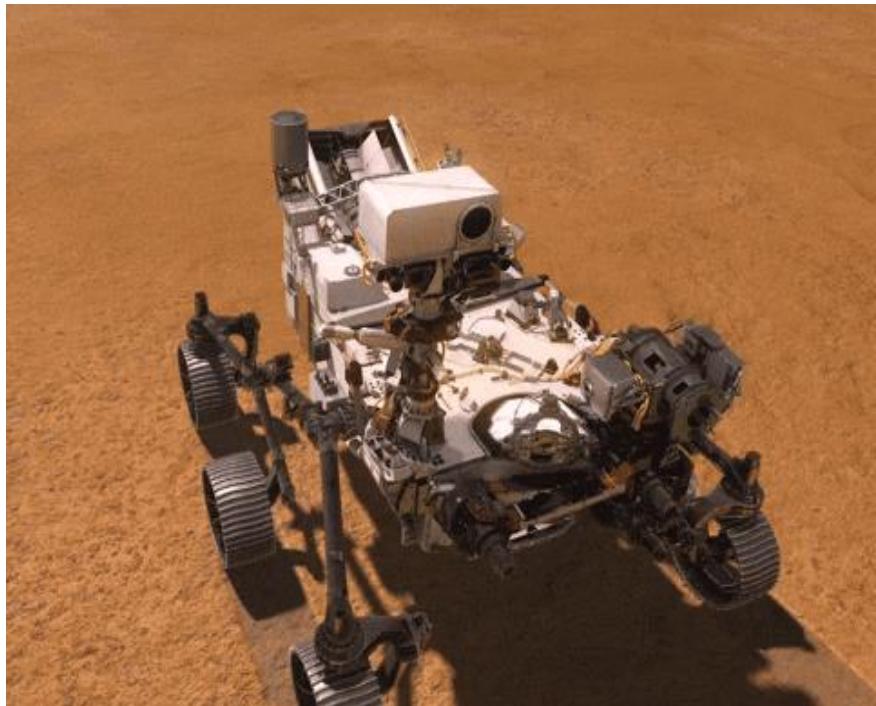
# Video

# What's next?

Anomalous and premature wheel wear

Caused by sharp rocks in Mars terrain

Wheel design was made according to the current knowledge

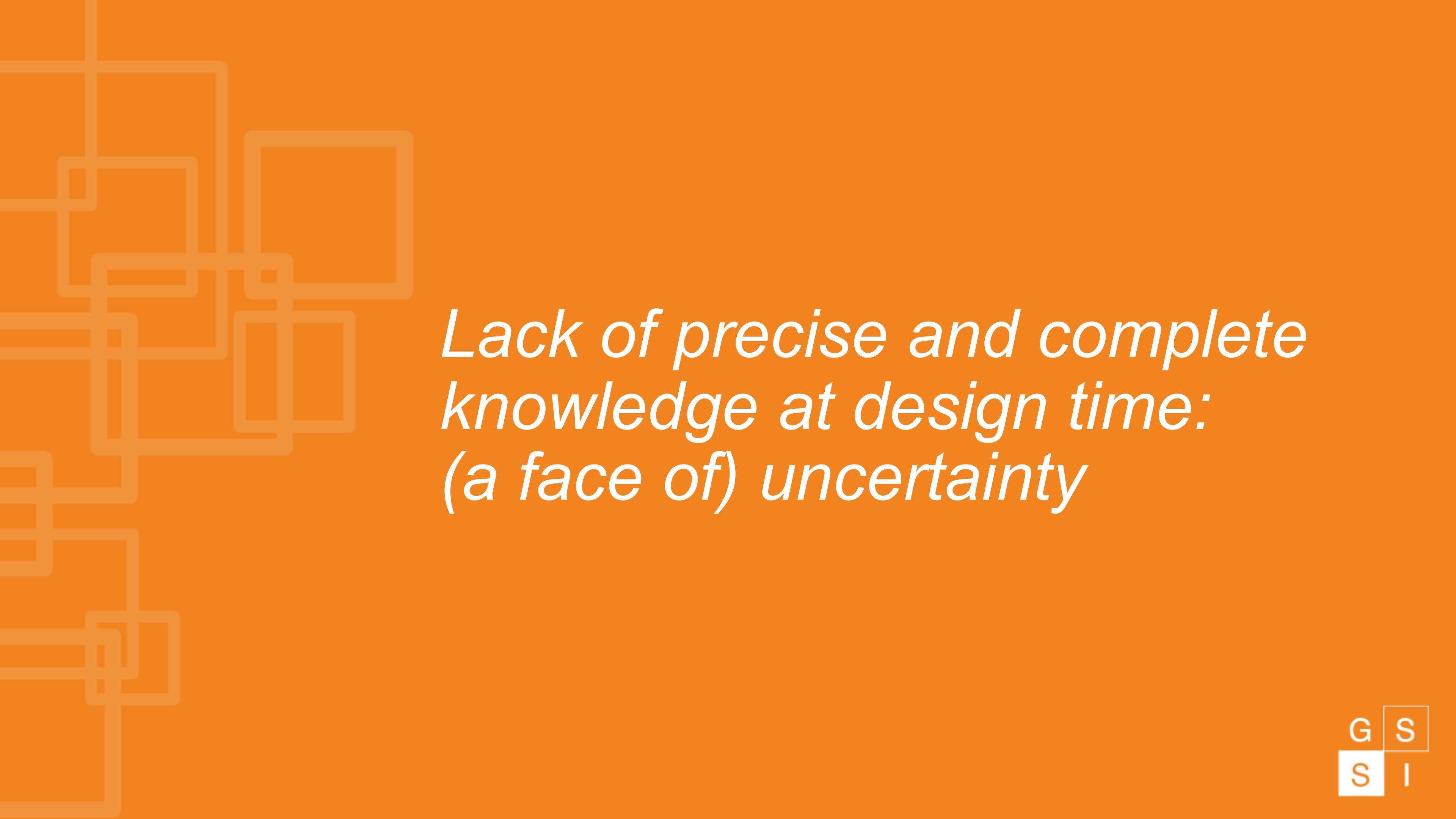


Engineers adapted navigation to solve the issue

Different navigation for different terrains

Required a software patch

NASA's MSL "Curiosity" rover issues



*Lack of precise and complete  
knowledge at design time:  
(a face of) uncertainty*

# Dealing with uncertainty in robotic missions

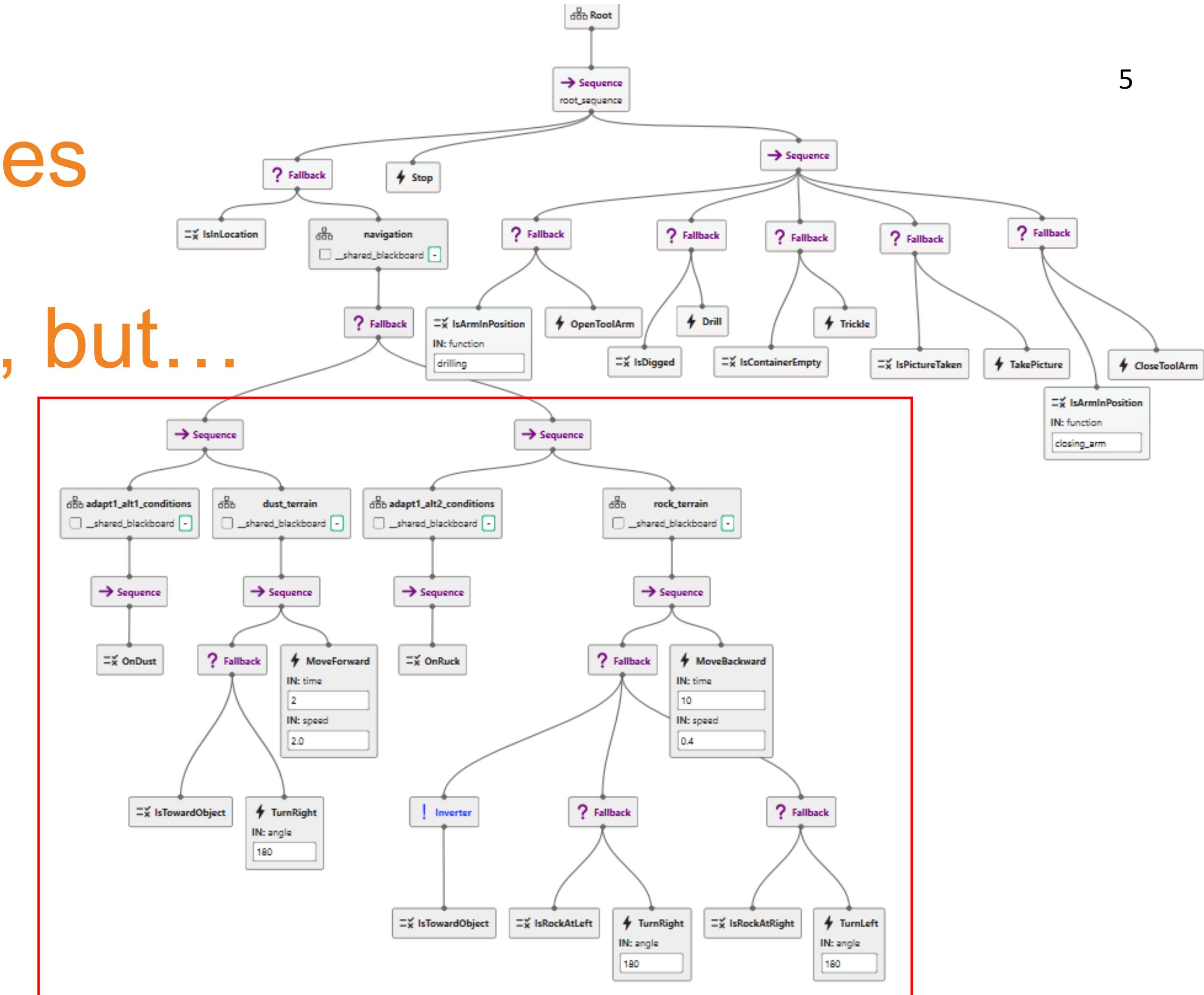
Effects of events/conditions may be unknown at runtime

Self-adaptation is needed to handle uncertainties at runtime

Impractical to specify all the alternative behaviors in a unique model

Sometimes it impossible (they are not known!)

# Behavior Trees enable reactiveness, but...



# Adaptable & Uncertainty-aware BTs

## Introducing *adaptable nodes*

Abstract nodes that model points of uncertainty

Manage *known-unknowns*

Placeholder for alternatives

## Goals

Avoid hard-coding of the alternatives

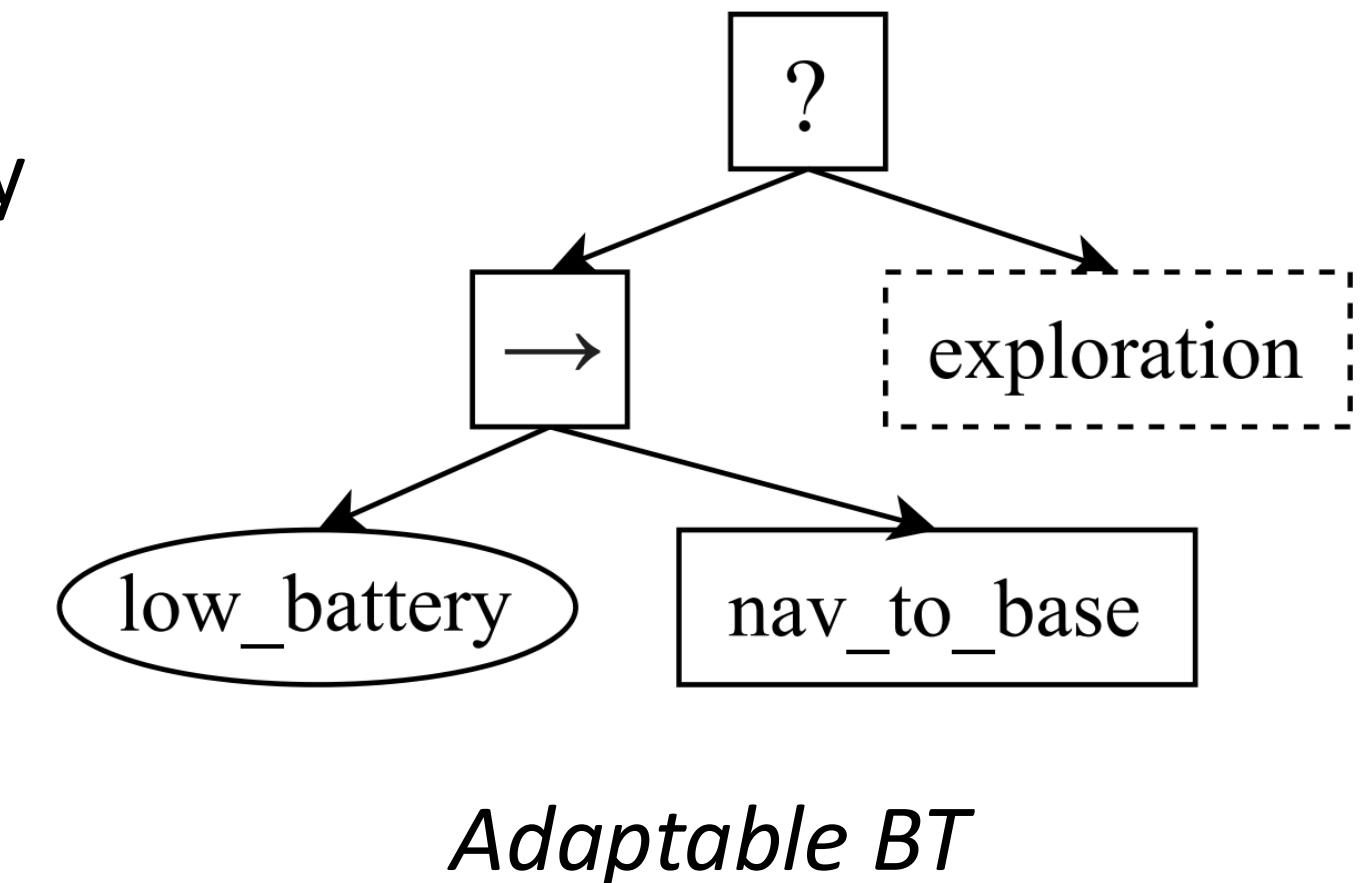
Increase modularity and flexibility

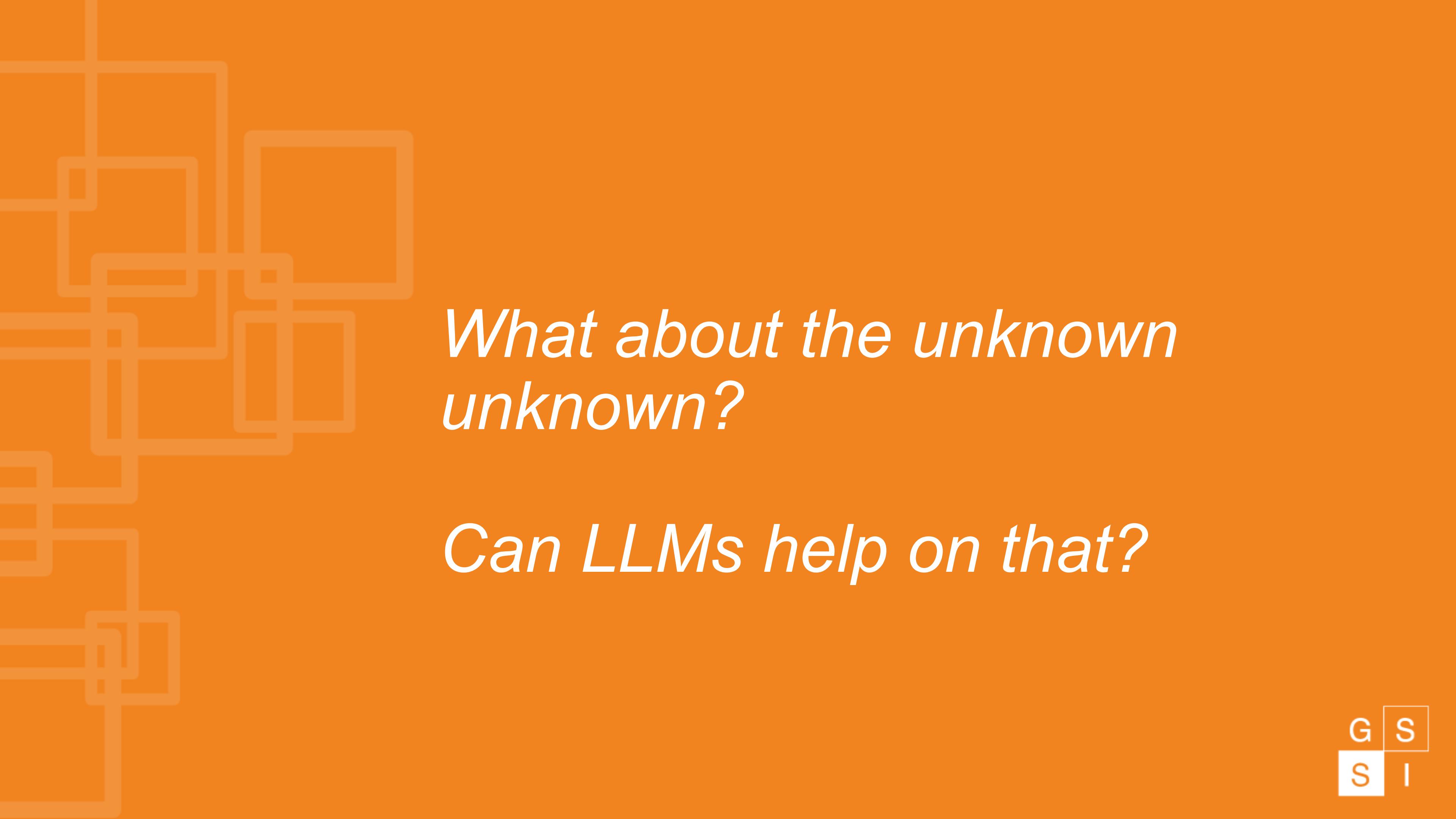
Allow behaviors addition/update

## What's missing?

Specification of the conditions for alternatives

Runtime support





*What about the unknown  
unknown?*

*Can LLMs help on that?*

## *Domain Specific Languages*

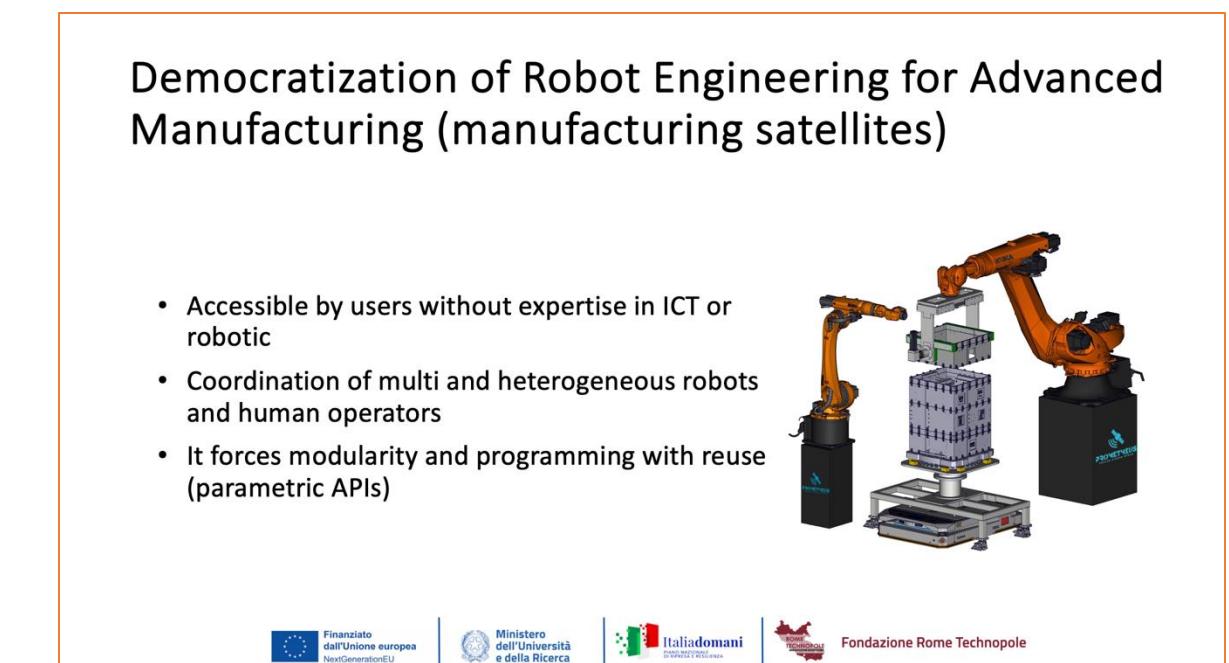
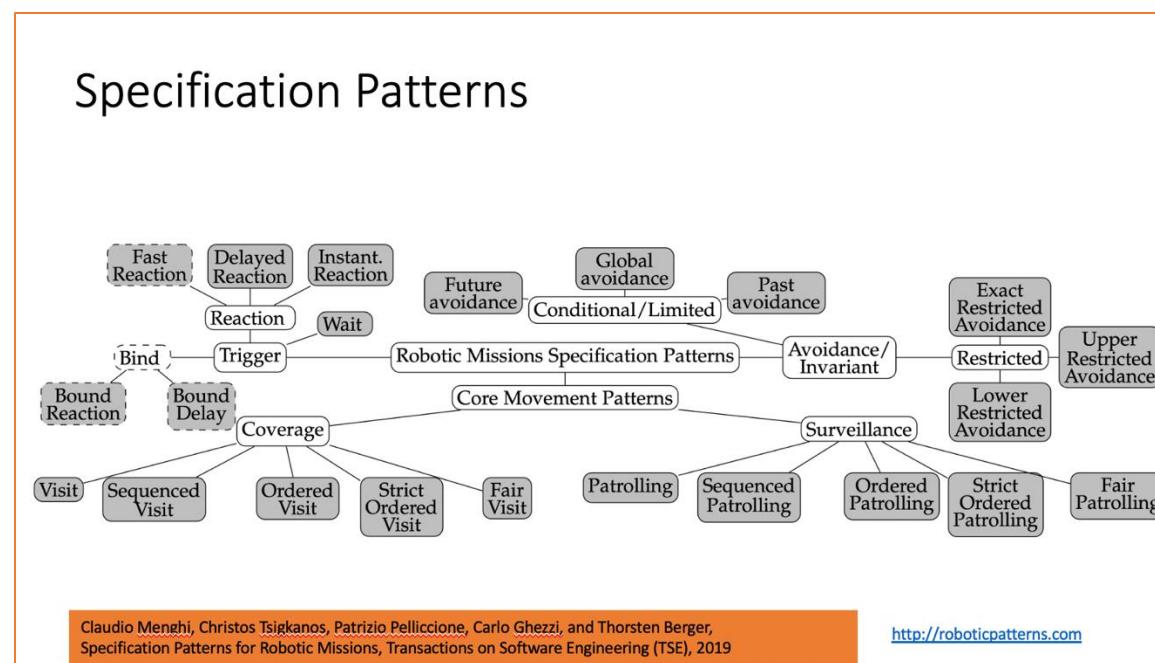
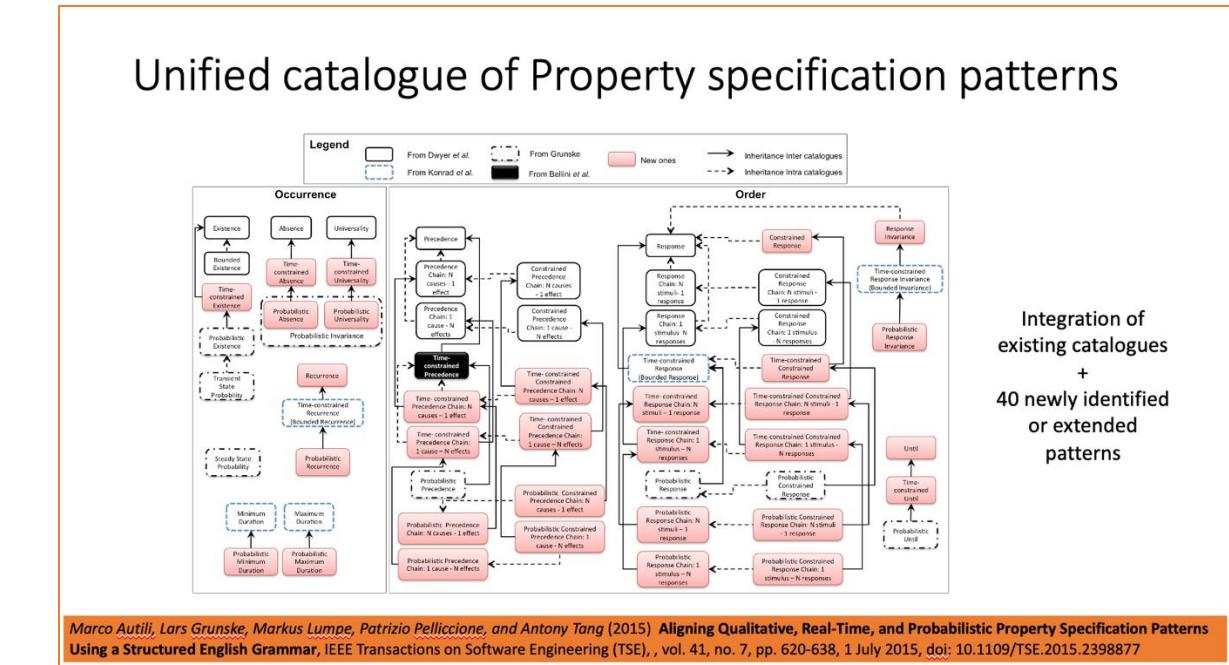
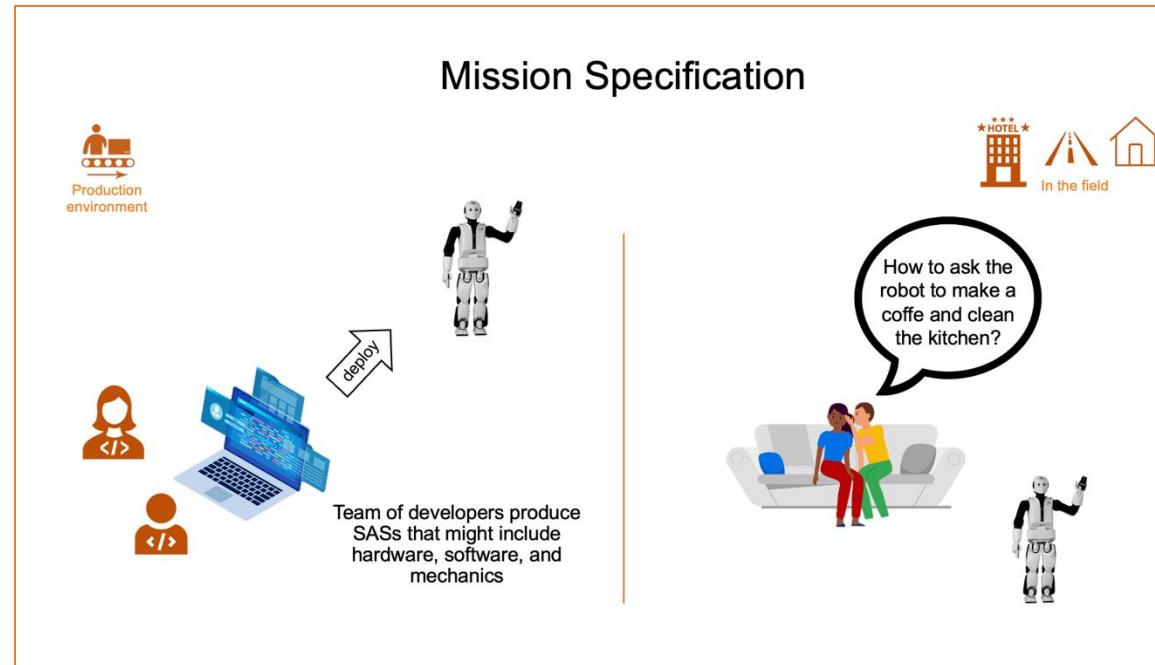
*Are the patterns domain specific?*

*We are not reusing them for the Smart factory*

*We probably need another step of abstraction  
in specific domains, like agriculture, space  
exploration, manufacturing*

*...plus patterns not focusing only on  
movements (in a map)*

# Summary





# GRAN SASSO SCIENCE INSTITUTE



Patrizio Pelliccione

[patrizio.pelliccione@gssi.it](mailto:patrizio.pelliccione@gssi.it)

<http://www.patriziopelliccione.com/>



[www.gssi.it](http://www.gssi.it)

