



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

www.gssi.it      

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](#)

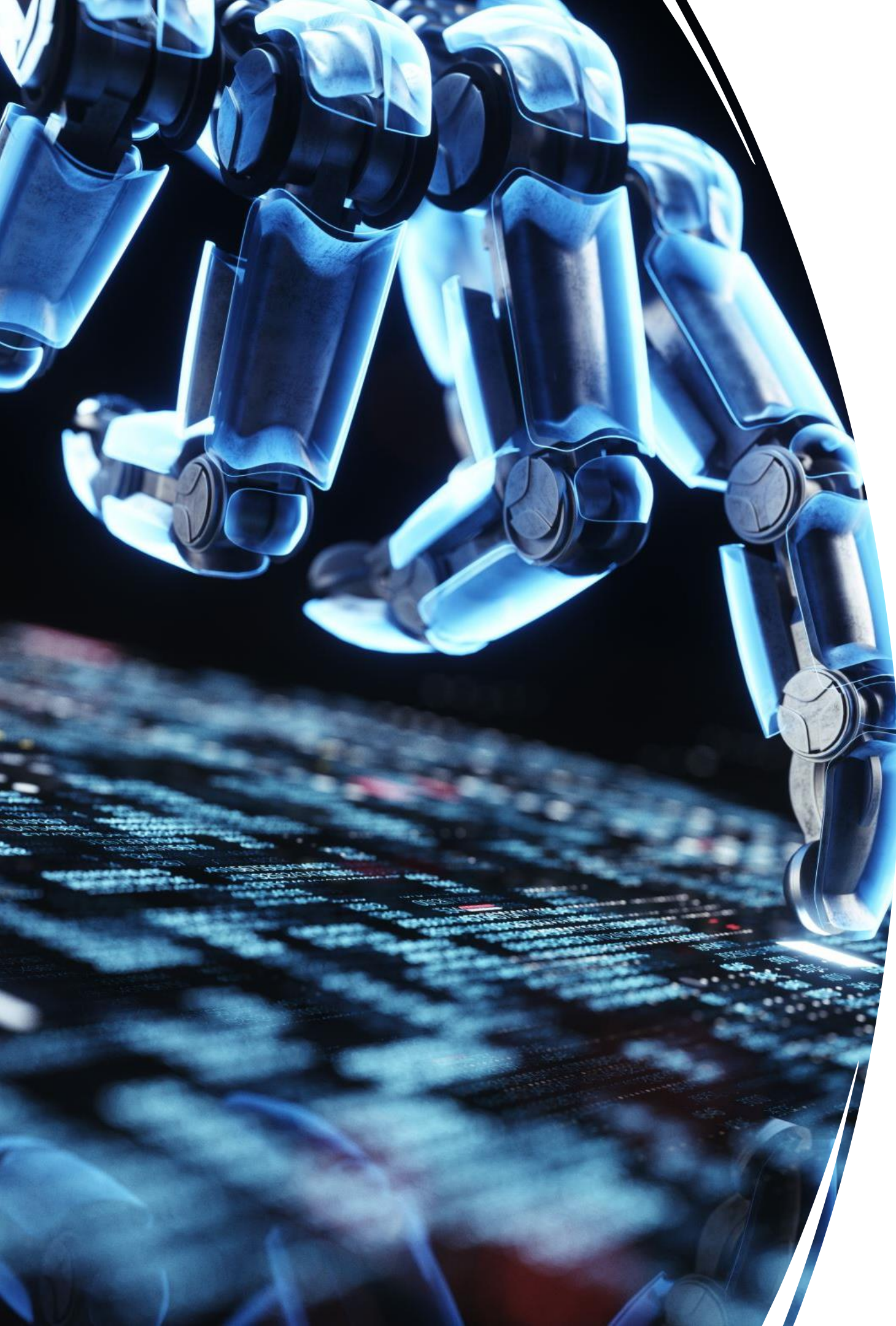


At the Tecnopolo Tiburtino hub in Rome, Thales Alenia Space's all-digital factory will employ advanced technologies for the production of satellites

- *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?

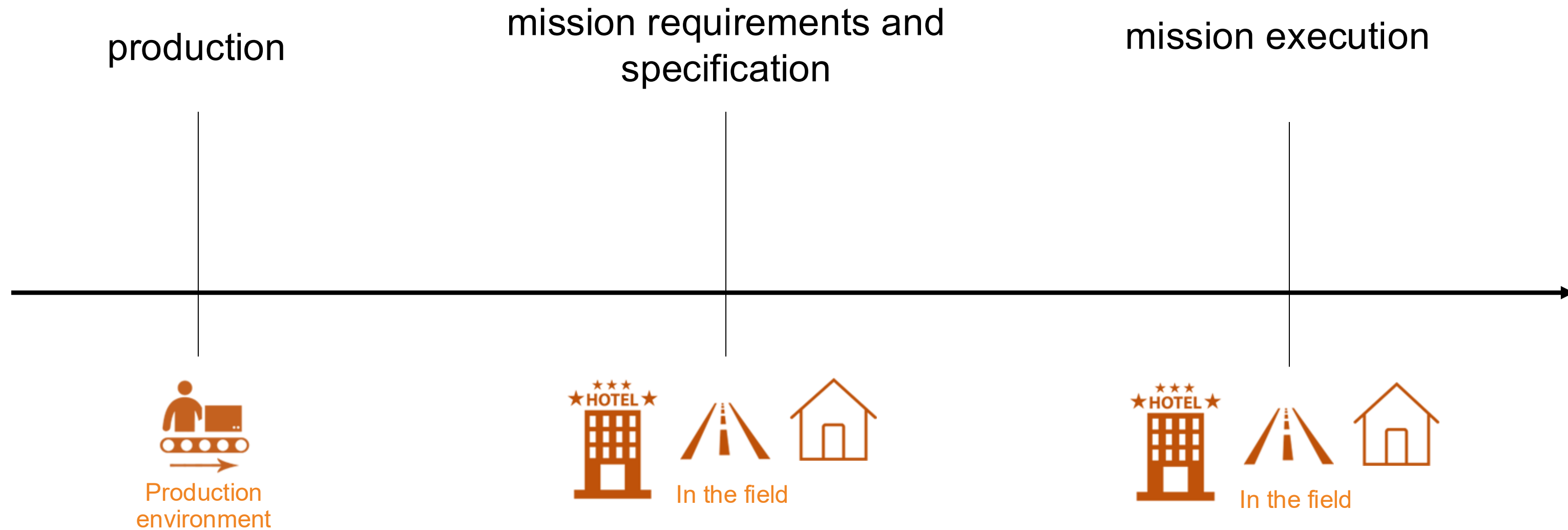
Different degree and complexity of interaction that get close to programming



Robotic Mission

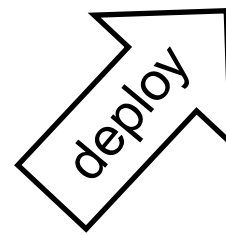
- A **mission requirement** describes the high-level tasks that a robotic software must accomplish.
- A **mission specification** is a formal and precise description of what robots should do in terms of movements and actions.
- **Robotic mission engineering** concerns expressing robotic missions in high-level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

Robots programming: beyond production environment

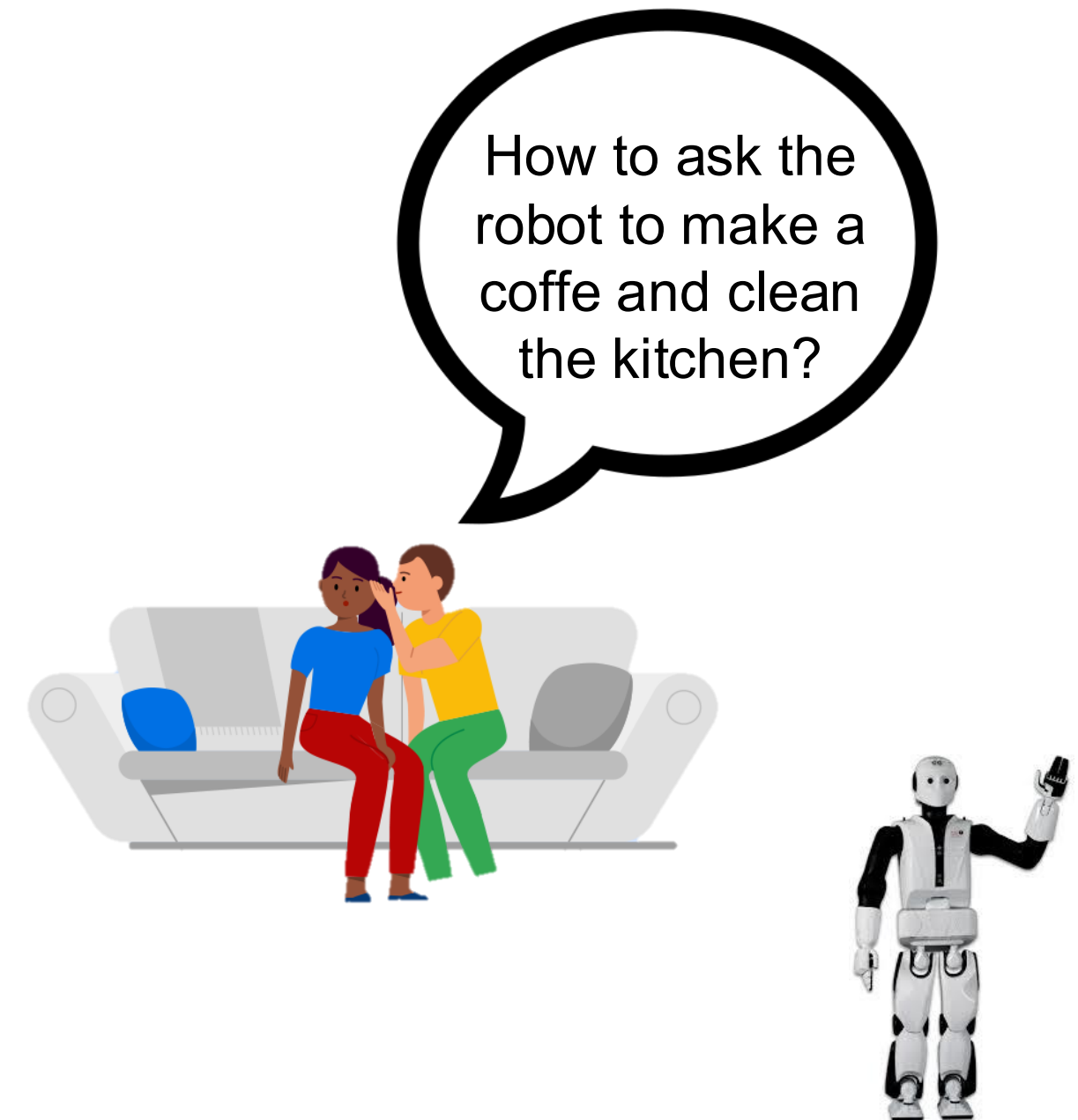


Different stakeholders

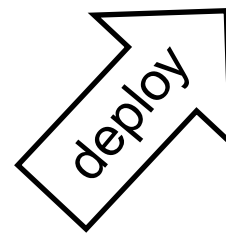
Mission Specification



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



Mission Specification



Programming extends in the field



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

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



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.

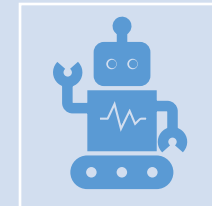
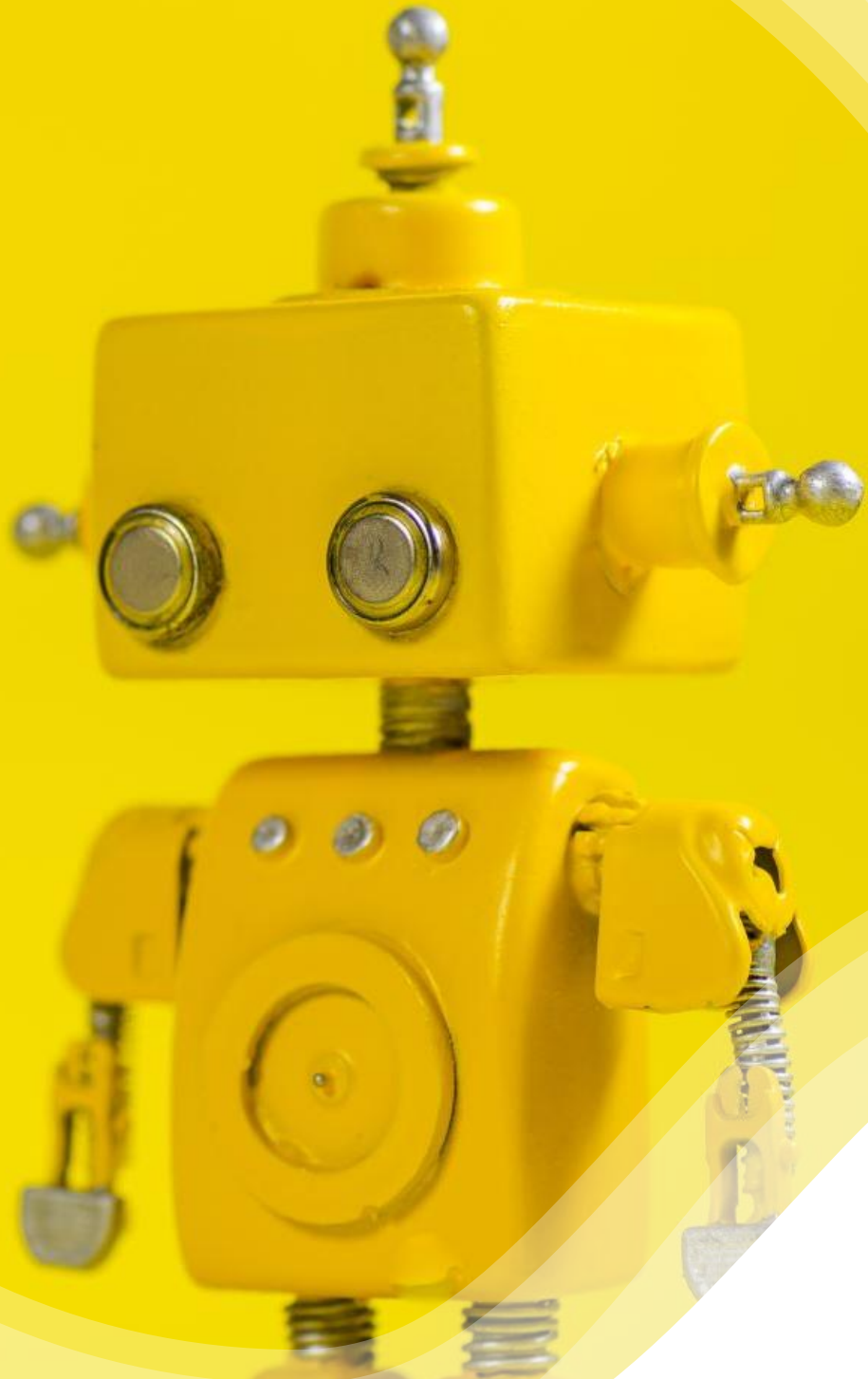
In addition, the robot should **perform regular check-ups** on people with particular conditions during free-time. It **needs to understand basic requests** and will alert the nurse in case of need, pose basic riddles or show simple pictures to test baseline human capabilities, ask the persons about 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.

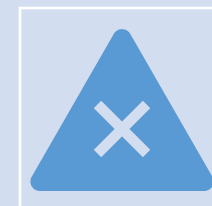
Exemplars: <https://github.com/Askarpour/RoboMAX>

Video: <https://www.youtube.com/watch?v=txZCcABkycQ>

Variability and uncertainty



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.

What we learn from
practitioners

Which type of variability?

		RQ1: Drivers of variability			
Environment Obs 1: Environment events Obs 2: Environment features Obs 3: Inclusion of humans		Robot Hardware Obs 4: Services and capabilities Obs 5: Hardware customization impact		Mission Obs 6: Expertise of human operators Obs 7: Human-robot interaction	
		Obs 8: Comparison in drivers of variability			
		RQ2: Variability management practices			
Strategies Obs 9: Installation process Obs 10: Scenario modelling Obs 11: Generic configurations	Mechanisms 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	Strategies 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	Mechanisms Obs 22: Middleware Obs 23: Certification standards Obs 24: Version control Obs 25: Reuse mechanisms Obs 26: Libraries	Strategies Obs 27: Generic missions	Mechanisms Obs 28: Mission-specification mechanisms
		Obs 29: Comparison in variability management			
		RQ3: Variability-Related Challenges			
Obs 30: Generic solutions Obs 31: Parametric configuration		Obs 32: Generic solutions among robots Obs 33: Testing variant-rich systems Obs 34: Integration and lack of standards Obs 35: Trade-offs		Obs 36: Mission specification Obs 37: User-friendly tools	
		Obs 39: Comparison in variability challenges			

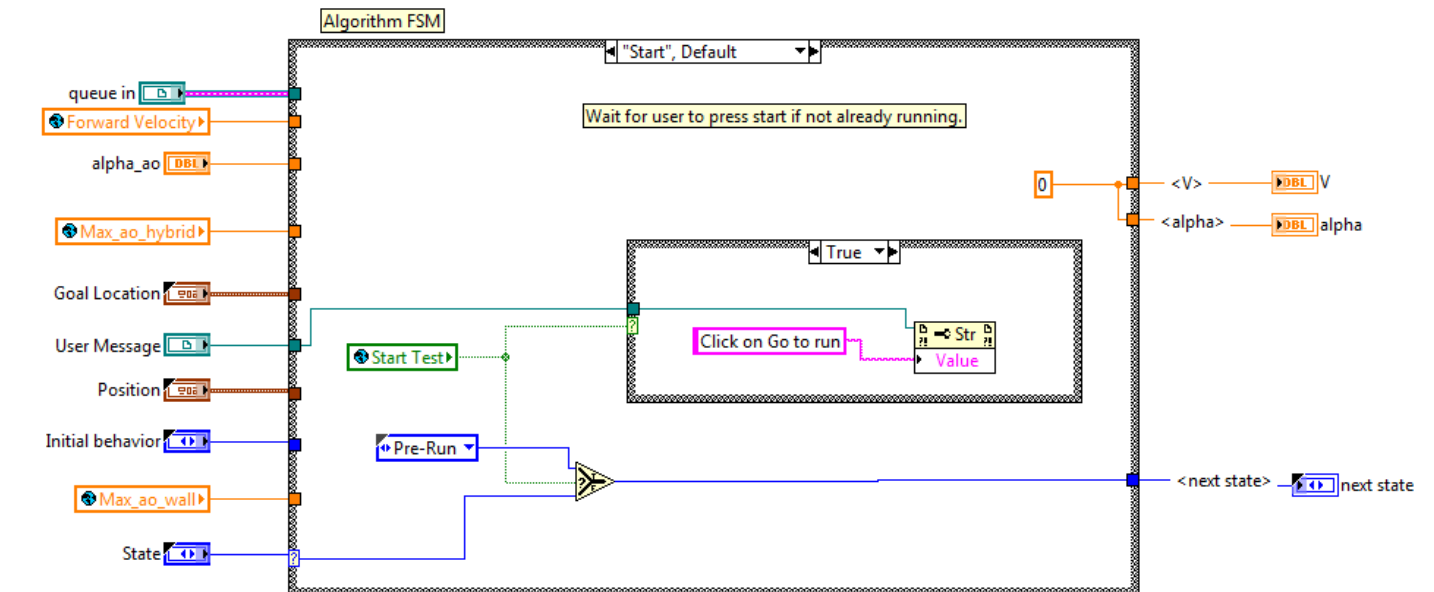
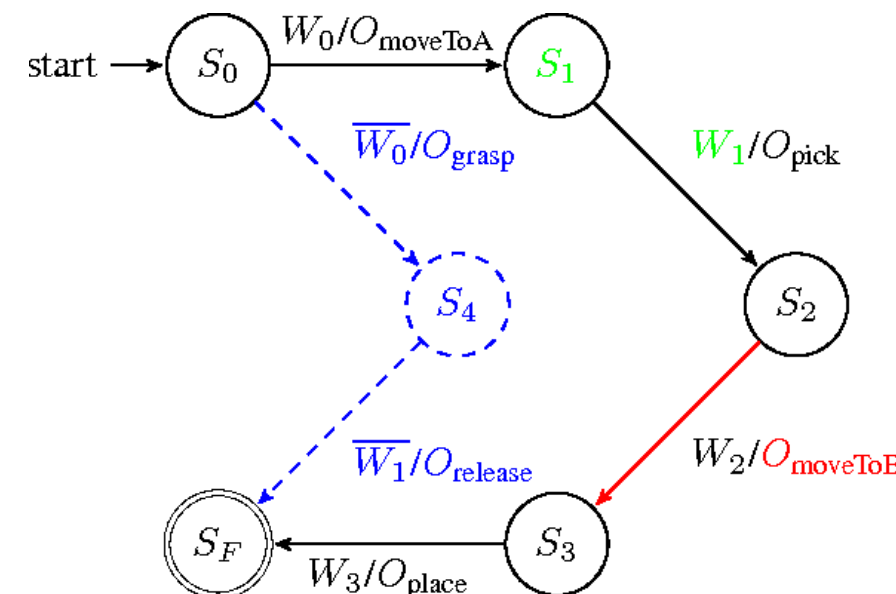
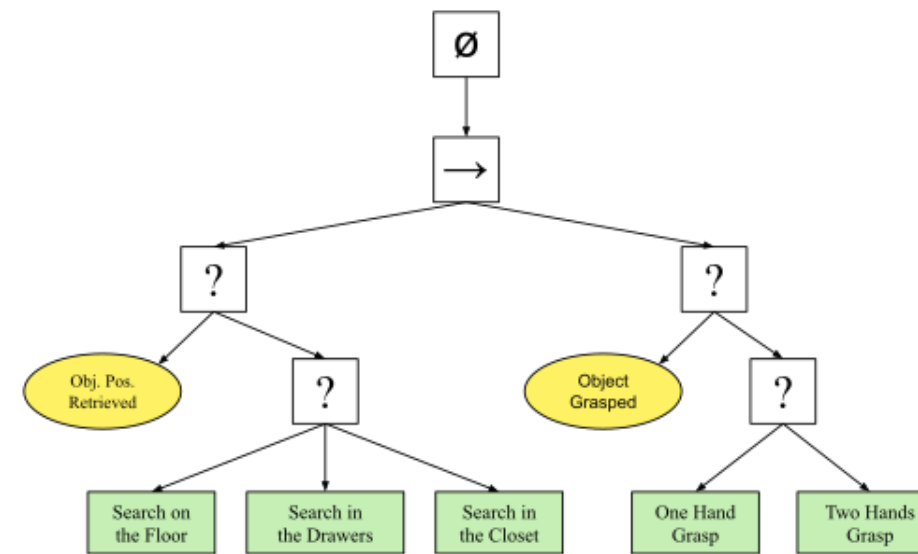
Potential failures

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

How to specify missions?



$$\Phi 1 = \langle \rangle ((r \text{ in } l1) \ \&\& \ \langle \rangle (r \text{ in } l2))$$



```

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.

```

Simple enough?

Means to specify robotic missions

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

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](#)



 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

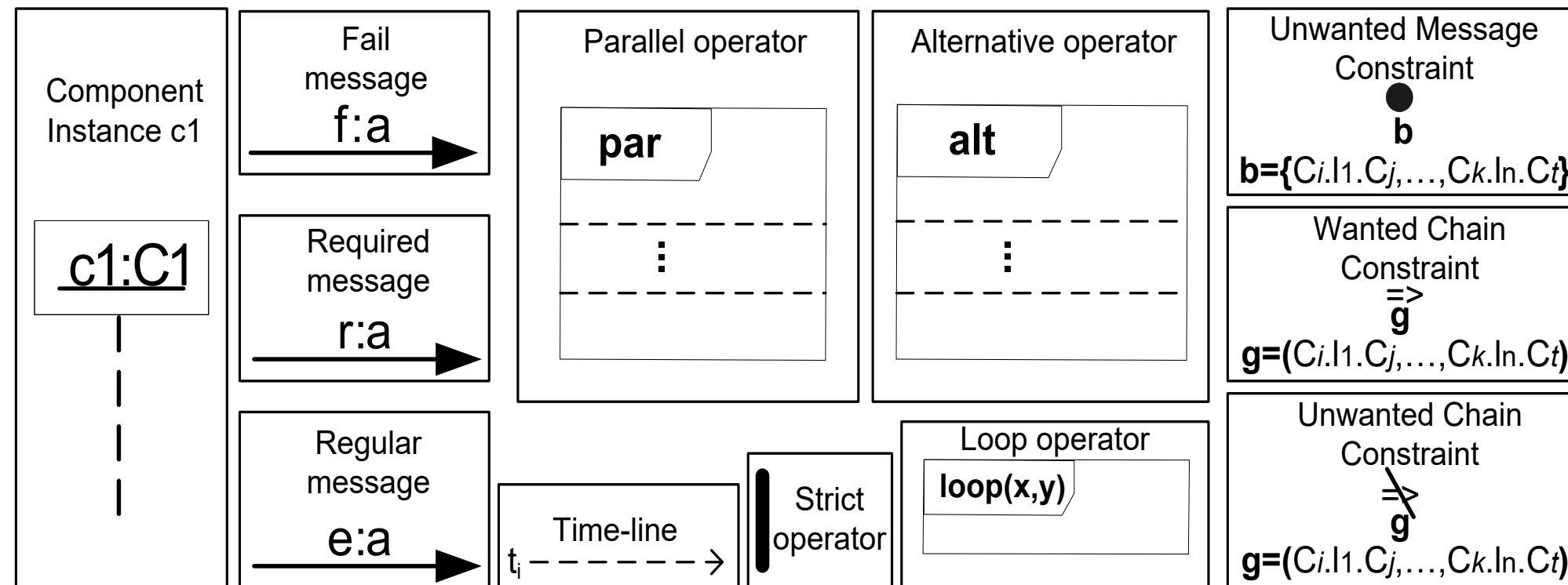
- Languages to facilitate the temporal properties specification
- Property Specification Patterns



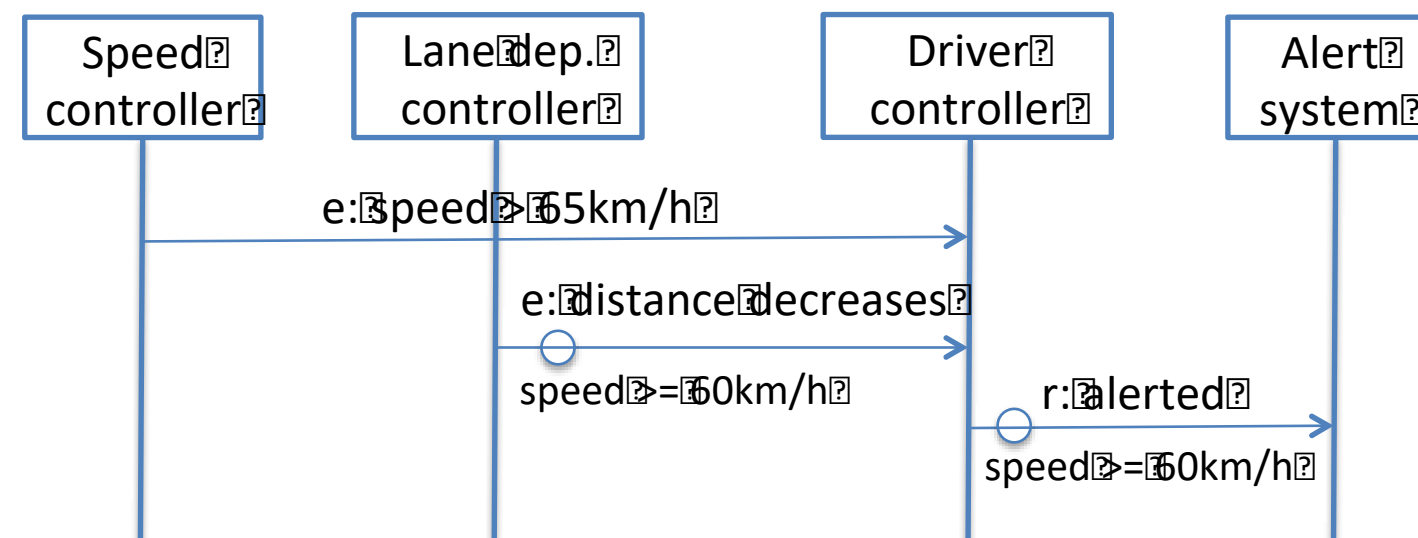
Main idea

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
 - 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.pragmadev.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

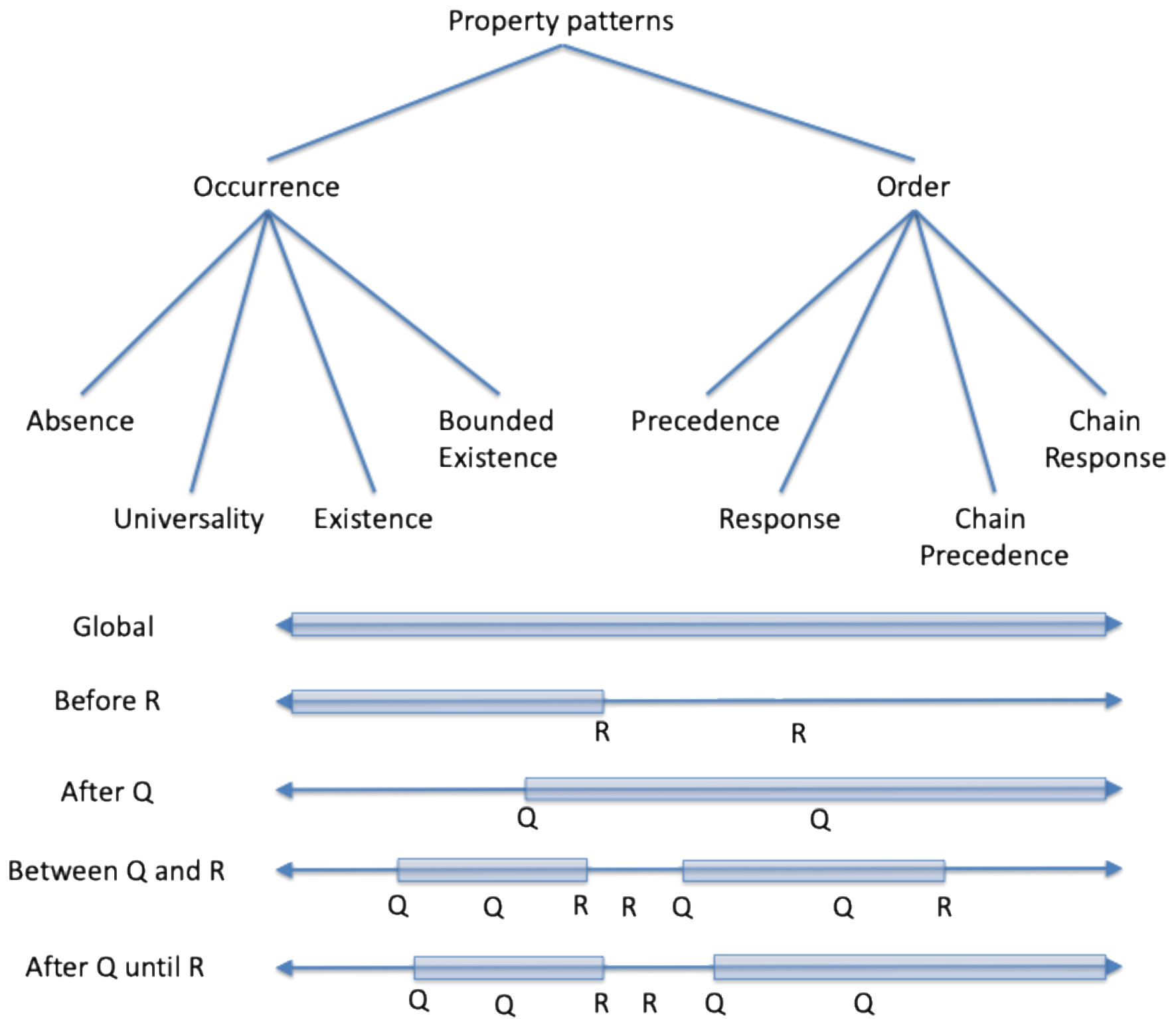
An example: Response pattern

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.

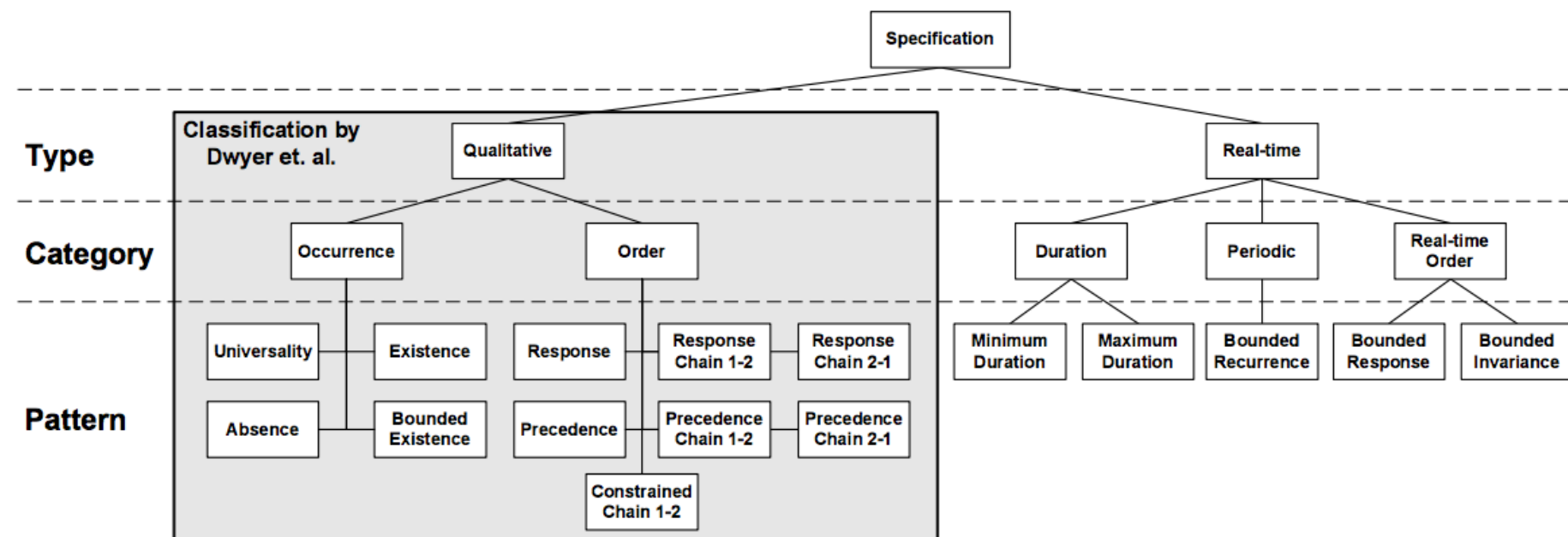
Also known as **Follows** and **Leads-to**.

S responds to P :

Globally	$[] (P \rightarrow \langle \rangle S)$
(*) Before R	$\langle \rangle R \rightarrow (P \rightarrow (!R \cup (S \ \& \ !R))) \cup R$
After Q	$[] (Q \rightarrow [] (P \rightarrow \langle \rangle S))$
(*) Between Q and R	$[] ((Q \ \& \ !R \ \& \ \langle \rangle R) \rightarrow (P \rightarrow (!R \cup (S \ \& \ !R))) \cup R)$
(*) After Q until R	$[] (Q \ \& \ !R \rightarrow ((P \rightarrow (!R \cup (S \ \& \ !R))) \cup R))$

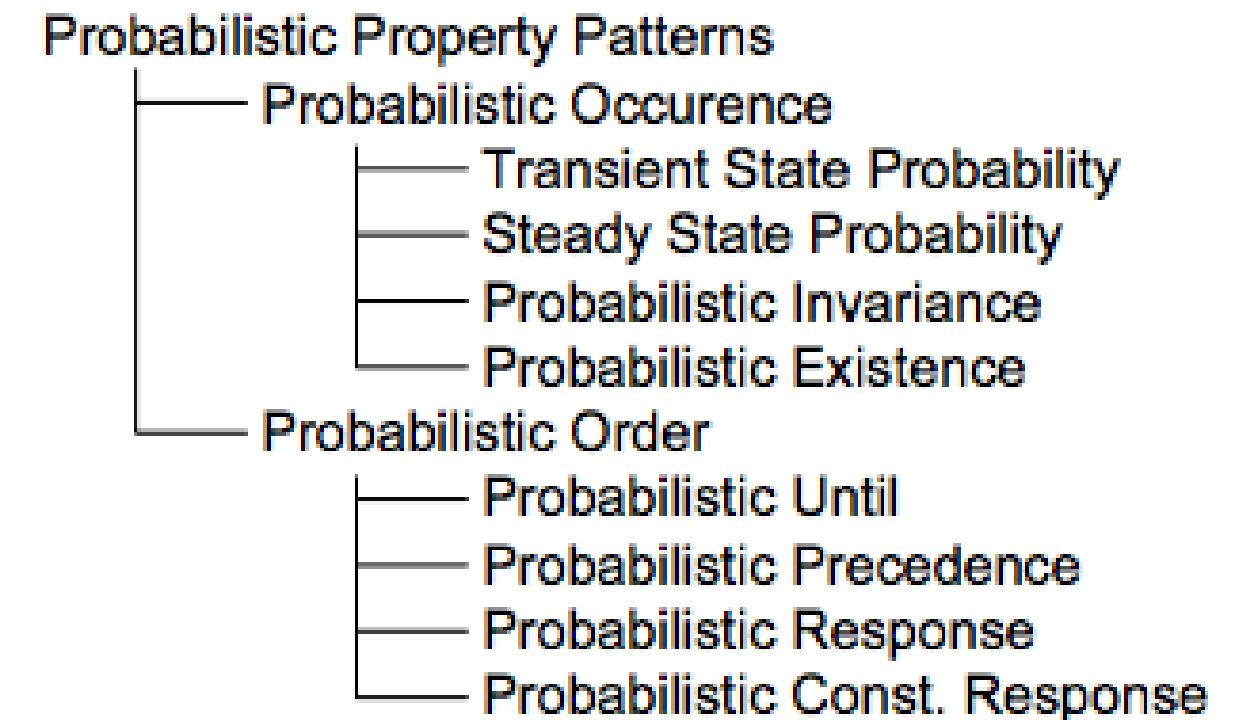


Real-time specification patterns



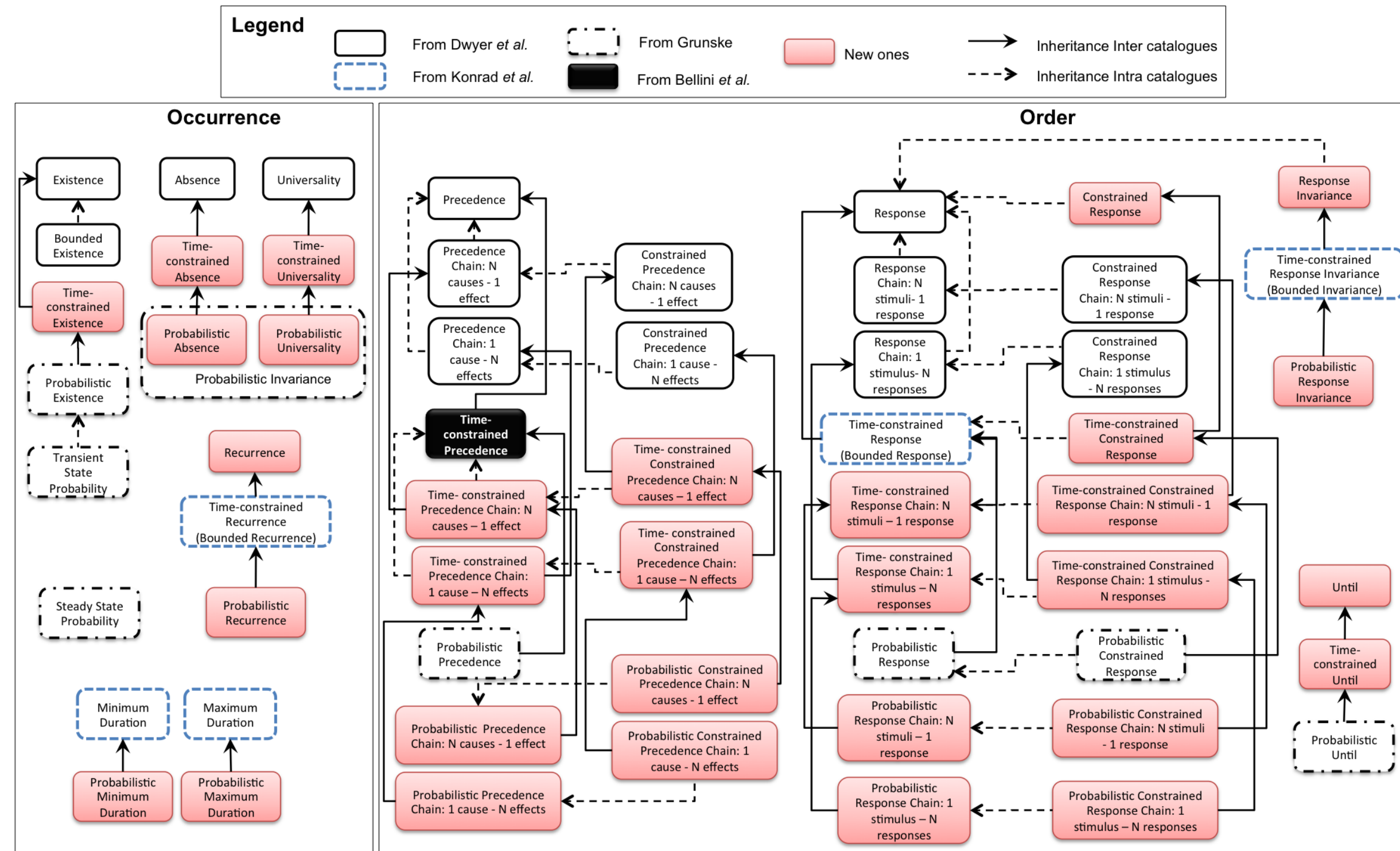
Sascha Konrad and Betty H. C. Cheng. 2005. **Real-time specification patterns**. In *Proceedings of the 27th international conference on Software engineering (ICSE '05)*. ACM, New York, NY, USA, 372-381.

Probabilistic Property patterns



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

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

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 l_1 and l_2 in this order”

l_1 and then l_2

Is it possible to visit l_2 before l_1 and then to visit l_2 ?

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 $l2$?

$\Phi1 = \langle \rangle (r \text{ in } l1) \ \&\& \ \langle \rangle (r \text{ in } l2)$ **vs.** $\phi2 = \phi1 \ \&\& \ ((!r \text{ in } l2)U(r \text{ in } l1))$

Intuitive and Simple **but also** Rigorous

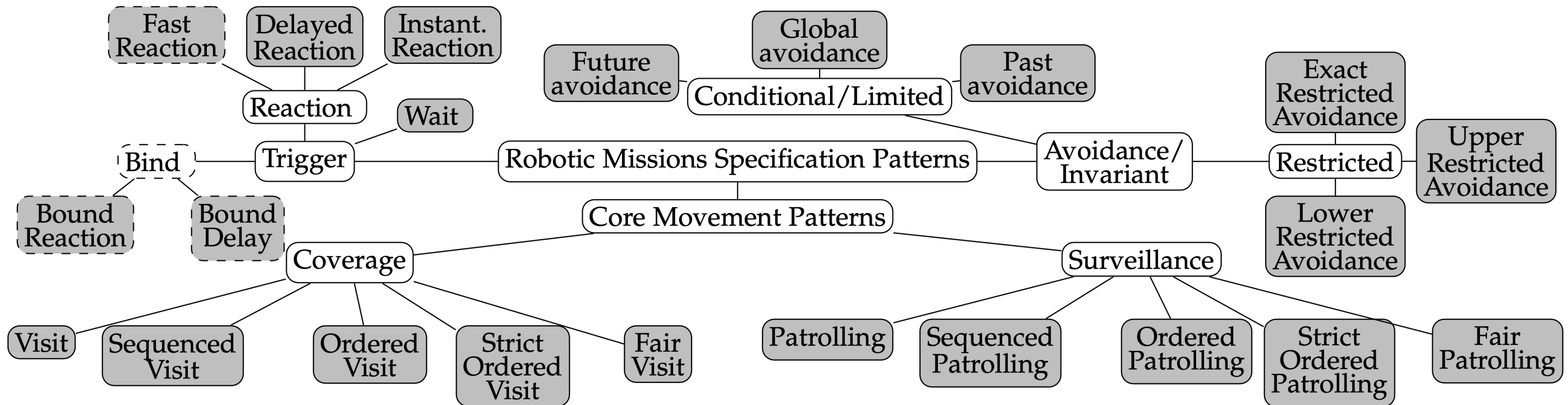
If we allow an ambiguous specification,
which behavior will have the robot,
and who will decide it?

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

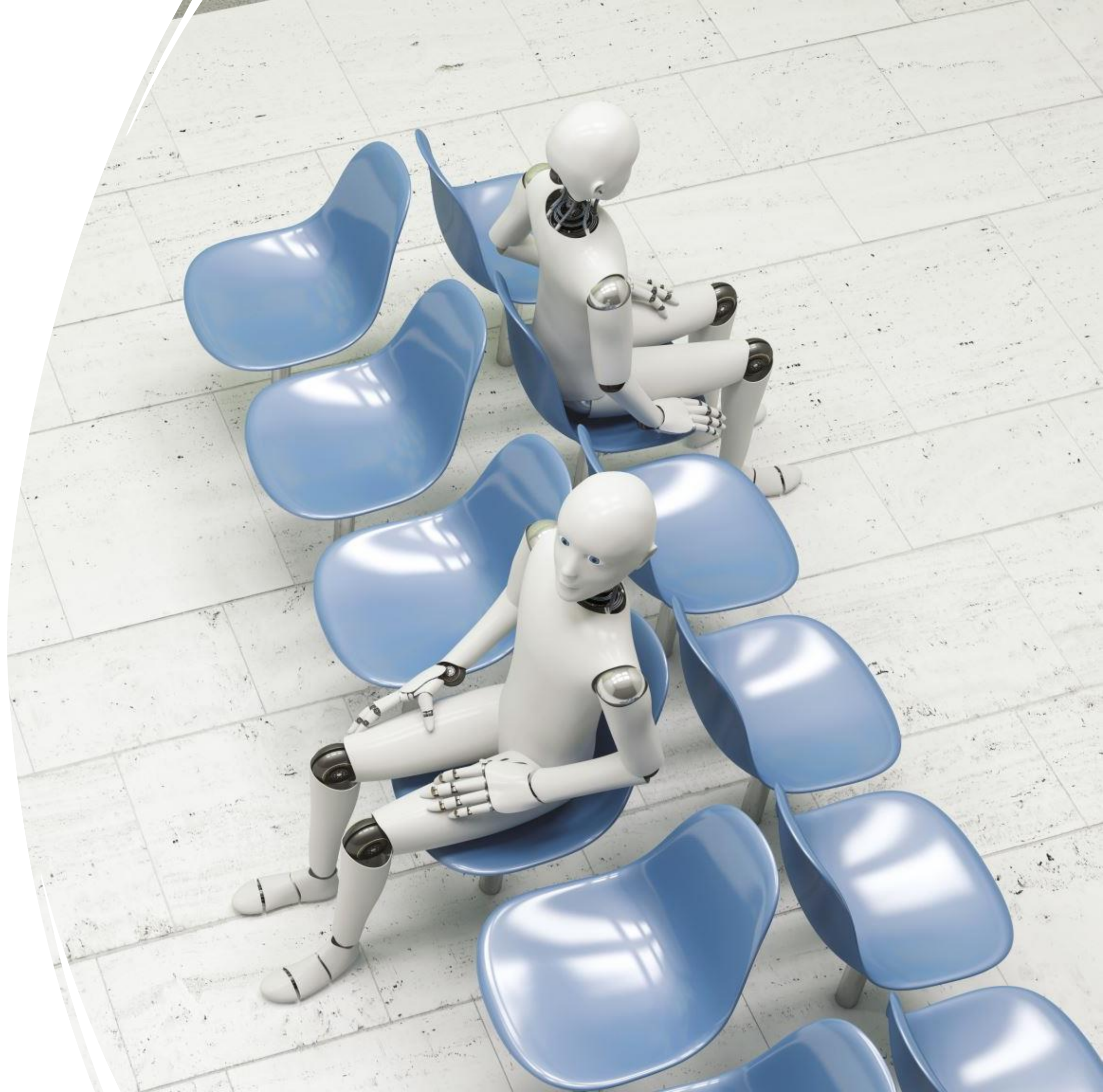
Are these patterns enough?

Example of mission requirement

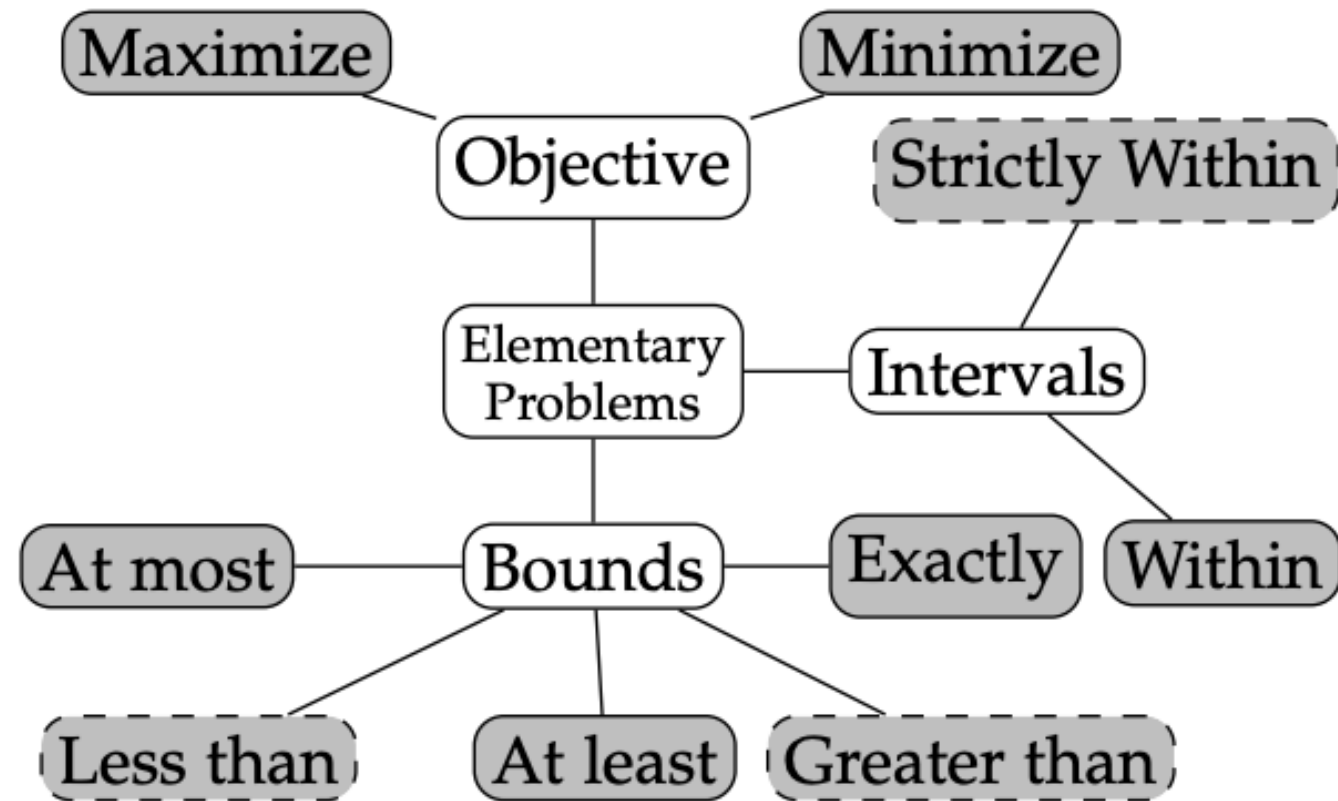
“After closure, the robots shall clean the electronics store. After cleaning, they shall visit a set of predefined store locations, each at least once, to record the items present on shelves after closure. The robots must **minimize the time required to perform this activity**. The robots should also 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

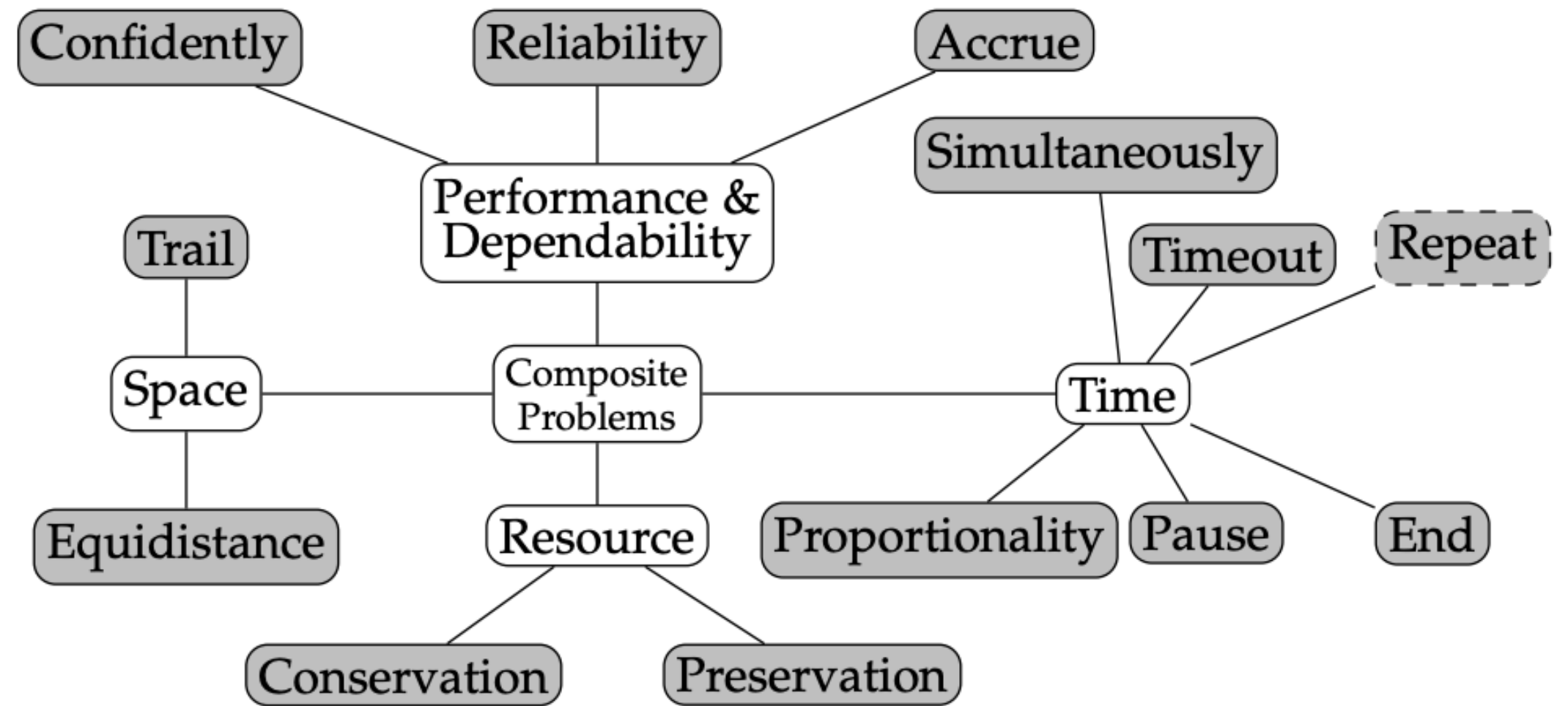
Users and operators of robotic systems often require behaviors that ensure quantitative constraints such as **upper bounds** on the **time** a robot takes to perform an action, the **energy consumption** to complete that action, or the **probability of failing** to achieve a mission goal.



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

New
ones

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

* miss, miss₁, miss₂ are missions; v, v₁, v₂ 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 $miss$.	maximize m $miss$
<i>Minimize</i>	Minimize m while performing the mission $miss$.	minimize m $miss$
<i>At most</i>	Keep m lower than or equal to v while performing $miss$.	m at most v $miss$
<i>Less than</i>	Keep m strictly lower than v while performing $miss$.	m less than v $miss$
<i>At least</i>	Keep m greater than or equal to v while performing $miss$.	m at least v $miss$
<i>Greater than</i>	Keep m strictly greater than v while performing $miss$.	m greater than v $miss$
<i>Exactly</i>	Keep m exactly v while performing $miss$.	m exactly v $miss$
<i>Within</i>	Keep m within the (closed) interval $[v_1, v_2]$ while performing $miss$.	m within v_1 and v_2 $miss$
<i>Strictly Within</i>	Keep m within the (open) interval (v_1, v_2) while performing $miss$.	m strictly within v_1 and v_2 $miss$
<i>Conservation</i>	Minimize the value of m performing $miss$.	conserve m while $miss$
<i>Preservation</i>	Keep the value of m within interval $[b_l, b_u]$ while performing $miss$.	preserve m within $[v_1, v_2]$ while $miss$
<i>Pause</i>	Pause the mission $miss$ for v time instants. Then, resume it.	pause v $miss$
<i>Timeout-deadline</i>	Execute $miss$. Stop the the execution when the timeout v is reached.	timeout v $miss$
<i>Repeat</i>	Repeat the mission $miss$ every v time units.	repeat $miss$ every v
<i>End</i>	Terminate mission $miss$ exactly at time v .	end $miss$ exactly_at v
<i>Proportionality</i>	Keep the time to perform $miss_1$ and $miss_2$ proportional by a factor v .	time of $miss_1$ proportional to [...]
<i>Simultaneously</i>	Execute the actions $act_1, act_2, \dots, act_n$ simultaneously.	execute rob actions $act_1, act_2, \dots, act_n$
<i>Accrue</i>	Maximize the performance m while performing $miss$.	rob accrue m while $miss$
<i>Reliably</i>	Ensure that the measure m is higher/lower than the value v .	achieve $miss$ with reliability m [...]
<i>Confidently</i>	Achieve $miss$ and ensure that confidence m is higher/lower than v .	achieve $miss$ with confidence m [...]
<i>Equidistance</i>	rob performs $miss$ by keeping rob_1 and rob_2 at the same distance.	rob $miss$ equidistance rob_1 rob_2
<i>Trail</i>	rob follows object o keeping a distance v .	rob trail o with distance v

* $miss, miss_1, miss_2$ are missions; v, v_1, v_2 are values; rob is a robot, o 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)

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

Further info about specification patterns

QUARTET: QUANTITATIVE ROBOTIC SPECIFICATION PATTERNS

PATTERN CATALOG

QUARTET DSL

QUARTET TOOL

REQUIREMENTS COLLECTION

EVALUATION

AUTHORS



QUANTITATIVE SPECIFICATION PATTERNS FOR ROBOTIC MISSIONS

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](#).

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

Claudio Menghi, Christos Tsigkanos, Mehrnoosh Askarpour, Patrizio Pelliccione, Gricel Vazquez, Radu Calinescu, and Sergio Garcia

Abstract—With many applications across domains as diverse as generic healthcare and agriculture, service robots are increasingly being deployed. Nevertheless, the designers of these robots often struggle with specifying their tasks in a way that their human counterparts can understand. This is a long-standing problem in the design of robotic systems. Recent research has addressed this problem by the introduction of robotic mission specification patterns and the design of robotic mission specification languages. These patterns support the definition of robotic mission requirements, for instance, the possibility of a particular task being executed in a particular order. These patterns support the definition of robotic mission requirements, for instance, the possibility of a particular task being executed in a particular order. These patterns support the definition of robotic mission requirements, for instance, the possibility of a particular task being executed in a particular order.

Index Terms—Robotic Systems Engineering, Robotic Mission Specification, Quantitative Properties, Domain-specific Languages, Probabilistic Mission Computation Tree Logic.

1 INTRODUCTION

The engineering of robotic applications is a complex interdisciplinary activity. Similar to many other domains, robotics requires contributions from different yet interdependent engineering sub-domains. Robotics engineers build low-level programs that allow higher-level control, while software engineers develop higher-level software components executed by robots [1]. As such, there is a great need for software solutions that can support the multiple activities of the engineering process – from requirements elicitation to software development and validation, e.g. [2], [3], [4], [5], [6], [7]. Mission specification is among the most important of these activities, as it entails capturing the requirements of robotic applications in a precise manner and in a form suitable for automatic processing. Mission specification involves specifying what the robot should do and how it should do it, as well as the conditions under which it should do it. Due to this multifaceted role, mission specification represents one of the main challenges in engineering robotics software [8], [9].

Typically, the engineering of robotics software is bootstrapped by requirements described in natural language, which are then translated into precise mission specifications. Such a mission requirement describes the high-level tasks that a robotic application must accomplish [10]. To be executable, this description should use a notation that is high-level and unambiguously [11], [12]. At the same time, it should provide a representation and enable the automatic verification and synthesis of the robotic software by formally and precisely specifying what the robot should do in terms of movements and actions [13], [14], [15]. We use the term mission specification profile for the problem of (automatically) generating a mission specification from a domain requirement. The main steps of mission specifications are: (i) unambiguous communication of the mission within the engineering team developing a robotic application and to other stakeholders, (ii) verification, where the robotic software or behavior generated from a robotic system or its simulation are checked against the specification, and (iii) synthesis, where behaviors that precisely satisfy the specification are constructed.

Mission specifications are often expressed in domain-specific languages (DSLs), many of which have been proposed over the last decades [16], [17]. These DSLs are usually implemented in a high-level programming language, enabling the generation of code that can then be executed within simulators or on real robots [18], [19], [20], [21]. However, these languages are typically bound to specific types of robots, and support a limited class of missions. Moreover,

<https://roboticpatterns.com/quantitative>




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





**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

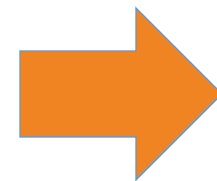
What kind of missions are specified in practice?

- 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


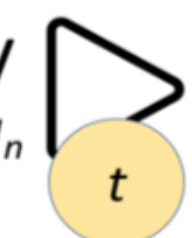
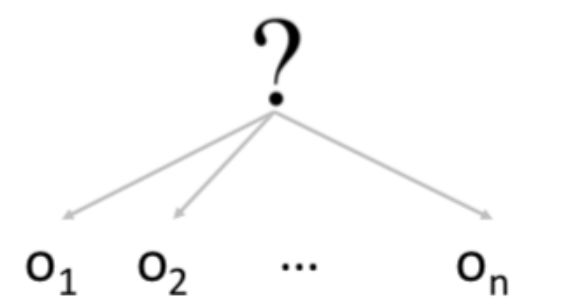
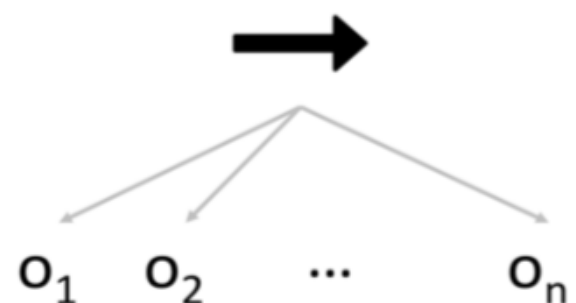
How to use these patterns to specify complex missions?

- 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

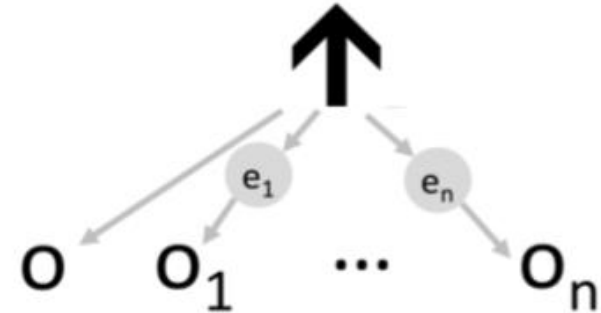
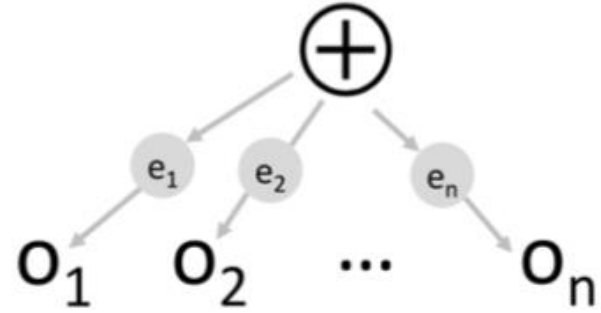
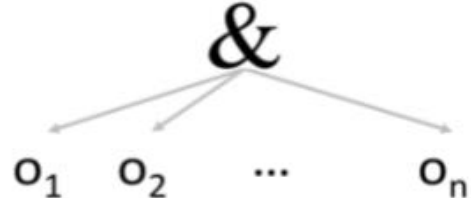
Domain Specific Language to specify missions

- 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.	$\{res_1, res_2, \dots, res_n\} = \{o_1, o_2, \dots, o_n\}$ if $(res_1 == \top \wedge \dots \wedge res_n == \top)$ then return \top else return \perp	 $parallel\{r1(o_1), \dots, rn(o_n)\}$	$r1[o1]$ $r2[o2]$ \dots $rn[on]$
Delegate $\Delta(\mathcal{E}, t)$	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).	execute (\mathcal{E}, t)	$l_1, l_2, \dots, l_n /$ a_1, a_2, \dots, a_n  $delegate(t \text{ locations } l_1, \dots, l_n)$ $delegate(t \text{ actions } a_1, \dots, a_n)$	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.	if $(\{o_1, o_2, \dots, o_n\} \neq \emptyset)$ then $res = o_1;$ if $(res == \perp)$ then $?(\{o_2, \dots, o_n\})$ else return \top else return \perp	 $fallback(o_1, o_2, \dots, o_n)$	$parent[fb]$ $fb_1[o1]$ $fb_2[o2]$ \dots $fb_n[on]$
Sequence $\rightarrow(\{o_1, o_2, \dots, o_n\})$	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.	if $(\{o_1, o_2, \dots, o_n\} \neq \emptyset)$ then $res = o_1;$ if $(res == \top)$ then $\rightarrow(\{o_2, \dots, o_n\})$ else return \perp else return \perp	 $sequence(o_1, o_2, \dots, o_n)$	$[o1,o2,\dots,on]$

Operators of the DSL

EventHandler $\uparrow (e_1, \dots, e_n, o, o_1, \dots, o_n)$	Executes a by default operator o . Once an event e_i occurs, executes operator o_i in response. Once the execution of o_i is finished, resumes the operator o . Returns success if the operator o succeeds and all the events that occurred during the execution of o are correctly handled.	<pre>res = ⊥; while(res ≠ ⊤) res = o; if(res == ⊤) then return ⊤ if(e_i == ⊤), then i = 1, ..., n resint = o_i; if(resint == ⊥), then return ⊥ res = resume(o); return res</pre>	 <pre>eventHandler(default(o) except e1 (o1) except e2 (o2)... except en (on))</pre>	parent[eh] eh_default[o] eh_e1[o1] eh_e2[o2] ... eh_en[on]
Condition $\oplus(\{e_1, \dots, e_n, o_1, \dots, o_n\})$	Evaluates the conditions from the first to the last. If the evaluation of one or more conditions is true, executes the corresponding operators. Returns \perp if an operation is not successful, i.e., either it fails or an event occurs. Returns \top when all the executed operations return \top .	<pre>if(e1 == ⊤) then res = o1 if(res == ⊥) then return ⊥ ... if(en == ⊤) then res = on if(res == ⊥) then return ⊥ return ⊤</pre>	 <pre>condition(if e1 then (o1) if e2 then (o2)... if en then (on))</pre>	parent[cond] cond_e1[o1] cond_e2[o2] ... cond_en[on]
TaskComb. $\&(\{o_1, o_2\})$	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.	<pre>res = o1 && o2 && ... on if(res == ⊤) then return ⊤ else return ⊥</pre>	 <pre>combination(o1 and o2 and ... on)</pre>	[o1 && o2 && ... on]


```

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 )+ ')';

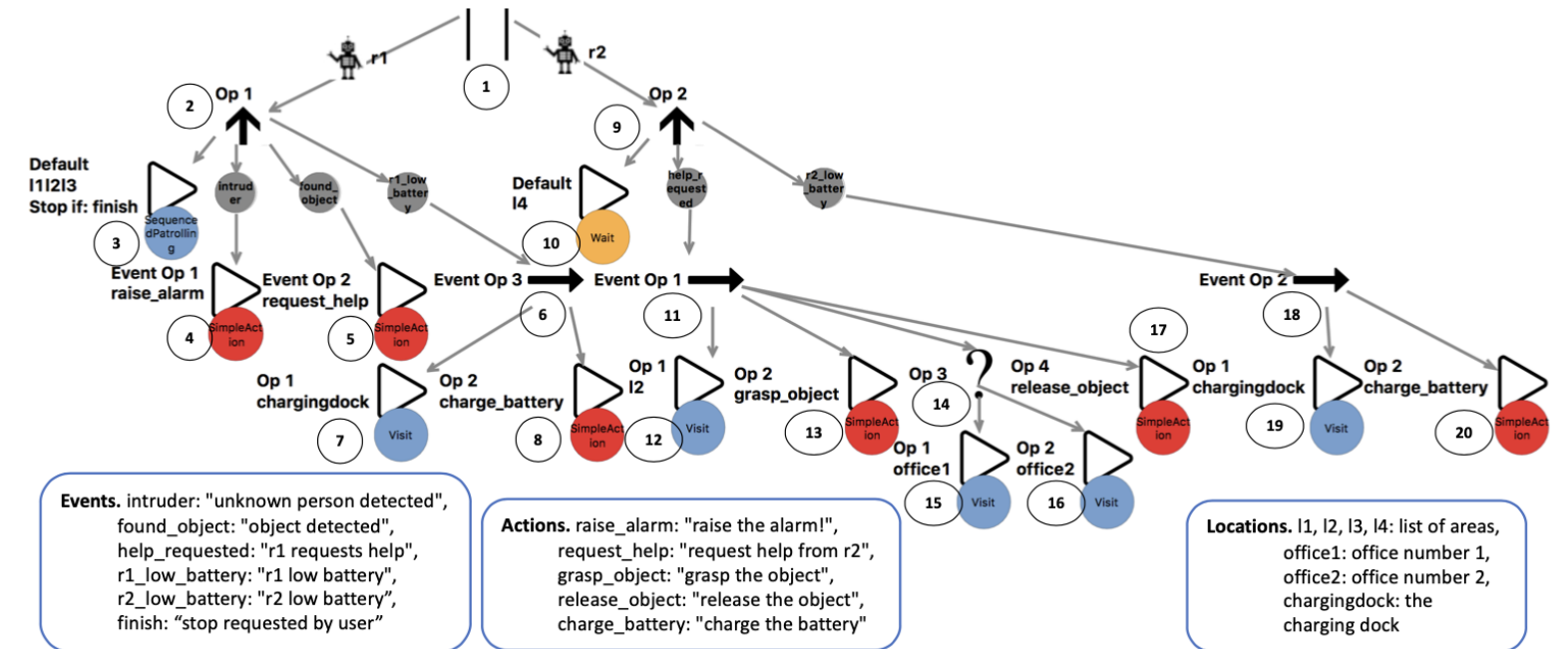
TaskCombinationOp:
'combination' '(' inputOperators+=Operator
(('&' | 'AND' | 'and') inputOperators+=Operator)+ ')';

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)



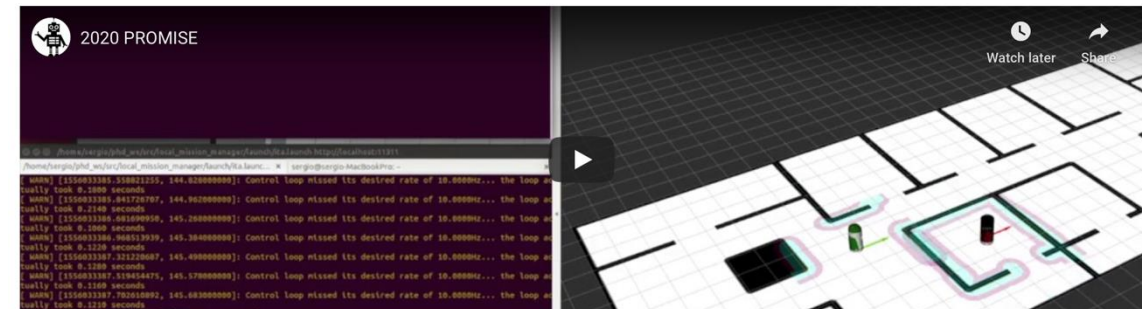
```

operators{ parallel{
  r1(eventHandler(
    default(delegate( SequencedPatrolling locations l1, l2, l3
      stoppingEvents finish))
    except intruder (delegate (SimpleAction actions raise_alarm))
    except found_object (delegate (SimpleAction actions request_help))
    except r1_low_battery (sequence(
      delegate(Visit locations chargingdock),
      delegate(SimpleAction actions charge_battery)))))
  r2(eventHandler(
    default(delegate(Wait locations l4))
    except help_requested( sequence(
      delegate(Visit locations l2),
      delegate(SimpleAction actions grasp_object),
      fallback (
        delegate(Visit locations office1),
        delegate(Visit locations office2)),
      delegate ( SimpleAction actions release_object ) ) )
    except r2_low_battery(sequence(
      delegate(Visit locations chargingdock),
      delegate (SimpleAction actions charge_battery)))))
  }
}

```

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.

High-Level Mission Specification for Multiple Robots

Sergio García
Chalmers University of Technology
Gothenburg, Sweden
sergio.garcia@chalmers.se

Patrizio Pelliccione
Chalmers University of Technology
Gothenburg, Sweden
patrizio.pelliccione@chalmers.se

Therrien Berger
Chalmers University of Technology
Gothenburg, Sweden
therrien.berger@chalmers.se

Tomás Bures
Charles University
Prague, Czech Republic
bures@fd.cvut.cz

Chao Menghi
University of Luxembourg
Luxembourg, Luxembourg
chao.menghi@uni.lu

Abstract
Multi-robot systems are increasingly used in our everyday life to perform increasingly more complex tasks. A variety of languages have been proposed to support robot teams in the systematic development of robotic applications, ranging from logical languages with well-defined semantics to domain-specific languages that combine these advantages with a more intuitive syntax. The present PROMISE, a novel language that enables domain-specific mission specification at a high level of abstraction, allows the specification of complex missions in a more friendly way, while having well-defined semantics. Our solution is to provide users to specify high-level goals instead of a series of specific actions the robots should perform. The language contains a set of atomic tasks that can be executed by robots and a set of operators that allow the composition of these tasks in more complex missions. The language is supported by a declarative logic programming framework through a central and high-level abstraction that can be integrated within a variety of frameworks. We integrated PROMISE with a software platform providing functionalities such as mission control and planning. We conducted experiments to evaluate the correctness of the specification and execution of complex robotic missions with both simulators and real robots. We also conducted free user studies to assess the usability of PROMISE. The results show that PROMISE effectively supports users to specify missions for robots in a user-friendly manner.

Keywords
Multi-robot, domain-specific language, mission specification

ACM Reference Format
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). New York, NY, USA, 13 pages. [https://doi.org/10.1145/3321021.3321027](#)

1 Introduction
Future robotic applications will include general purpose robots that are designed for real-world use in a variety of domains of everyday life, as analyzed by the EU FP7 Future Multi-Agent Robotics (FMR) project. For example, a user may want to assign the following mission to a robot application: "Visit all of the rooms in the house, and, for example, prepare coffee in the kitchen and eat it in the living room during night hours." This can be performed either programatically by a General Purpose Language (GPL) or by a domain-specific language (DSL), or by using logical languages that allow to provide a declarative description of the robotic application should achieve [1, 11, 15, 16]. A declarative specification can be defined using languages including domain-specific languages, or Linear Temporal Logic (LTL) or Computation Tree Logic (CTL). These logics are increasingly used in the robotics community and are becoming standard tools for specifying robotic missions as they can be automatically processed by planners [1, 11, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24].

A planner is a software component that receives as input a model of the mission specification and derives the sequence of actions about what must execute. However, writing correct formulas in temporal logic requires knowledge of these syntaxes and semantics, which makes mission specification a cumbersome and error-prone task, even for experts [1, 15].

127

PROMISE: High-Level Mission Specification for Multiple Robots

Sergio García
Chalmers University of Technology
Gothenburg, Sweden
sergio.garcia@chalmers.se

Patrizio Pelliccione
Chalmers University of Technology
Gothenburg, Sweden
patrizio.pelliccione@chalmers.se

Therrien Berger
Chalmers University of Technology
Gothenburg, Sweden
therrien.berger@chalmers.se

Tomás Bures
Charles University
Prague, Czech Republic
bures@fd.cvut.cz

Chao Menghi
University of Luxembourg
Luxembourg, Luxembourg
chao.menghi@uni.lu

ABSTRACT
Multi-Agent Robotics (MAR)? According to this roadmap, this will require: (1) that robots must be more strongly integrated with business operations? (2) highly abstracted development of software? (3) languages for different agents operating with autonomous users? In fact, these requirements are fulfilled by two recent studies [9, 10], whose authors claim that previously specifying missions and transforming them to automatic programming are one of the main challenges in robotics research engineering. Therefore, the current state of robot mission specification includes not only the lack of standardization in the tool while keeping the requirements. At the same time, we strive to enable a rigorous specification that facilitates previously and automatically the mission the robot must perform. In addition to a recent study [9], existing solutions fail at providing a trade-off of these qualities.

In this paper, we present a framework that supports users in mission specification for multi-robot applications. The framework integrates PROMISE (simPle ROBot Mission SpEciFication), a language designed as a DSL. This framework permits the specification of complex missions (i.e., complete behaviors, including safety-critical events) while having powerful for mission knowledge of programming languages. Moreover, PROMISE builds on a catalog of mission specification LTL-based patterns [10], ensuring the correctness of the specification and therefore the rigor of the tool. The framework also integrates different components for the automatic mission generation, modeling, and interpretation and management.

Building example. We use as running example for a reference experiment from a European project¹ with which our DSL is developed in parallel to enhance different missions throughout the paper. The example represents two robots collaboratively working in an industrial warehouse. A mobile platform (1) the robot to deliver items to other stations and can stop the processing using specific patterns. (1) continuously evaluates if

1. INTRODUCTION
It is noticeable the increase of investment and inclusion of robotic robots in several market sectors (e.g., logistics, medical, or farming) in the next few years². As the number of deployed robotic robots (industrial, domestic, service) will likely have to interact with robots in their everyday life, language for mission specification is an essential requirement (e.g., a hotel, a hospital, or a warehouse) where teams of robotic robots are deployed to achieve tasks collaboratively. These tasks will be managed and specified in domain-specific and real-world users who may not be knowledgeable on programming languages, according to the H2020 Robotics

Video: [https://youtu.be/3D5upYQ20Q8](#)

¹ [https://www.eurostars.eu](#)

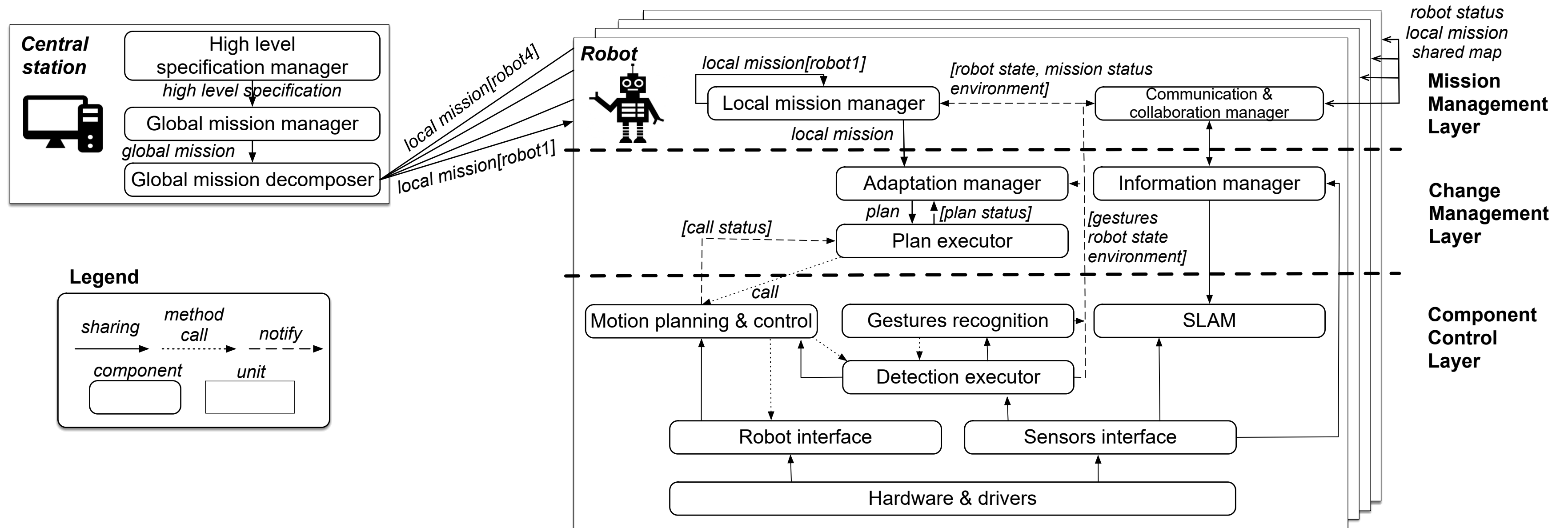
² [https://www.eurostars.eu](#)

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



SERA (Self- adaptive dEcentralized Robotic Architecture)

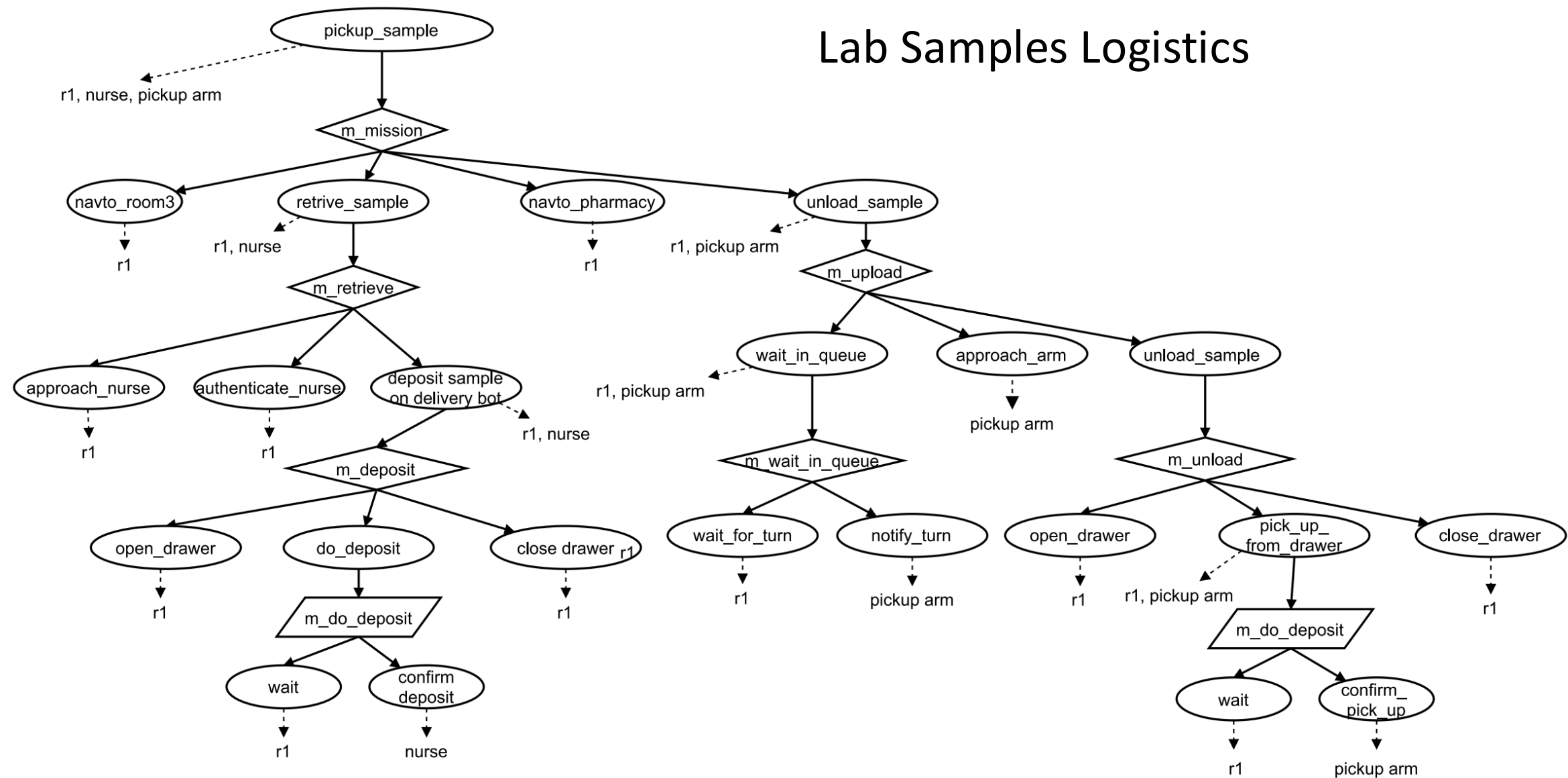


<https://co4robots.eu/>

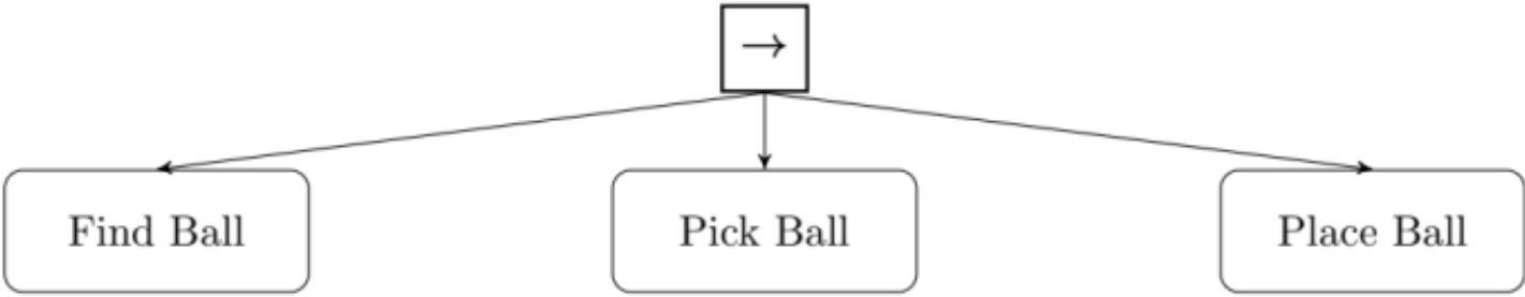
S. García, C. Menghi, P. Pelliccione, T. Berger and R. Wohlrab, "An Architecture for Decentralized, Collaborative, and Autonomous Robots," 2018 IEEE International Conference on Software Architecture (ICSA), Seattle, WA, 2018, pp. 75-7509.

Instantiated Hierarchical Task Networks (iHTN)

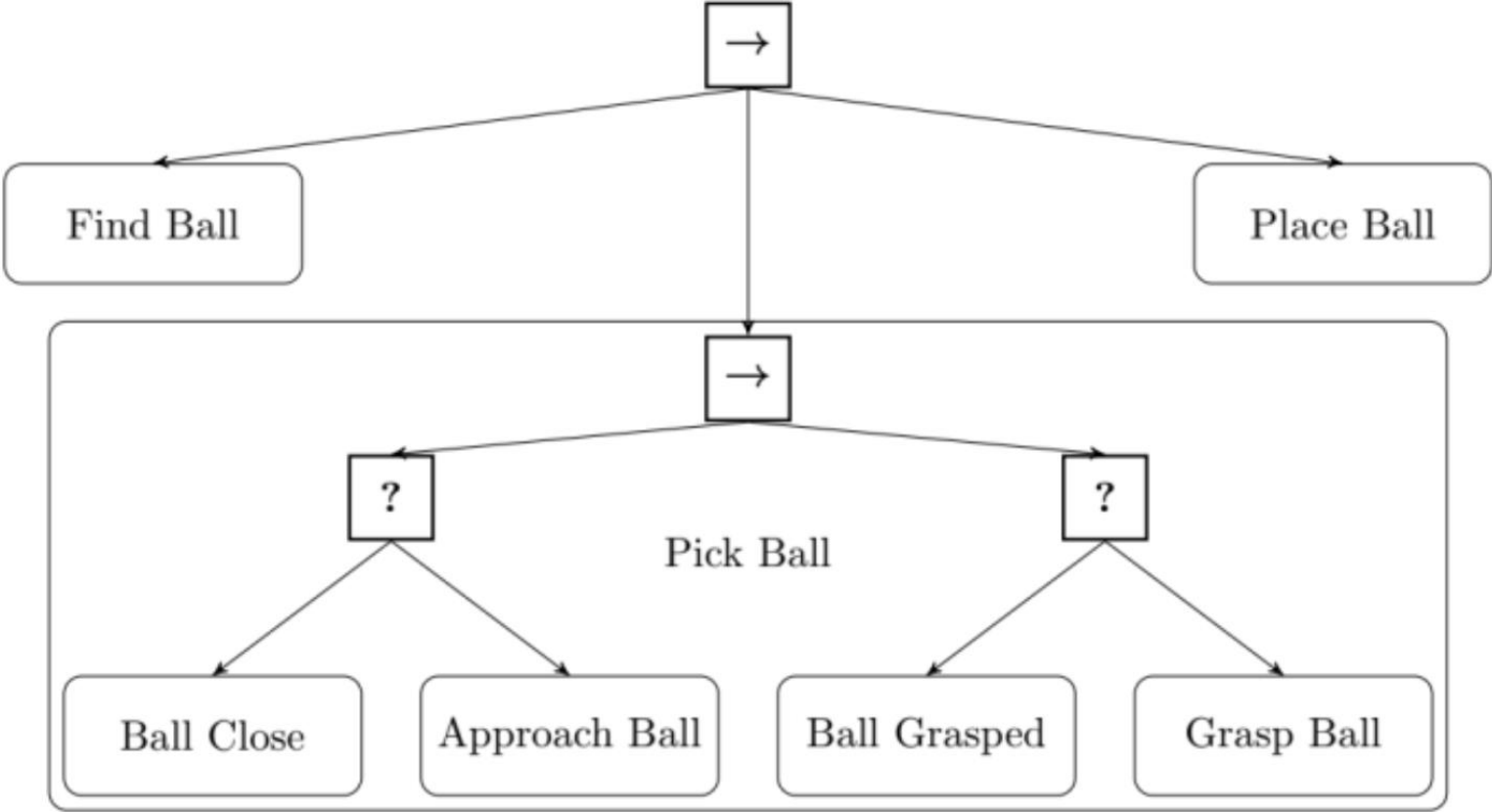
- Hierarchical Task Networks is a formalism for task planning.
- The Instantiated HTN (iHTN) formalism formalizes a multi-robot collaborative mission.
- Tasks (ellipses) are efforts that a set of agents (robots or humans) must undertake.
- A task can be abstract or concrete.
- Abstract tasks are refined 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).
- ...



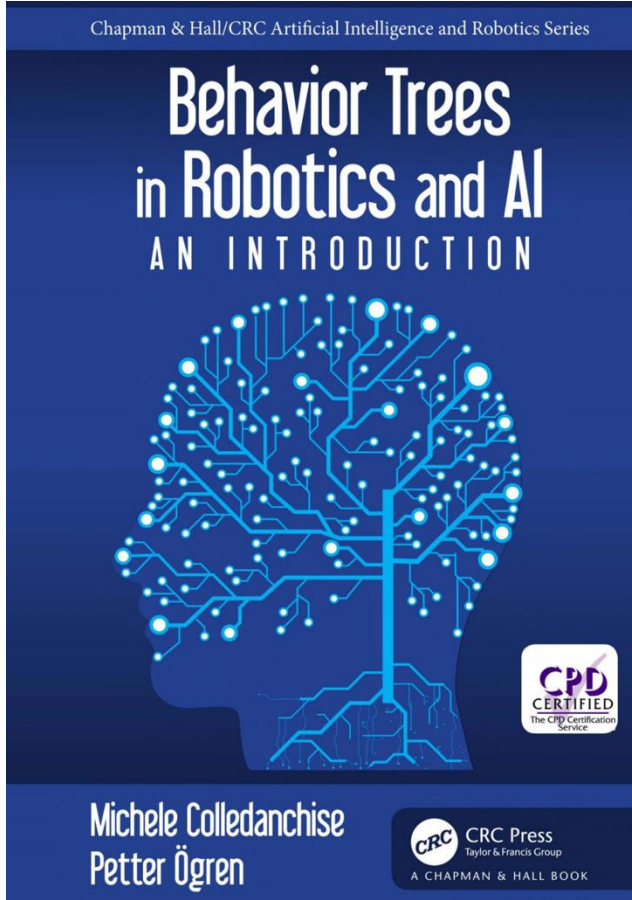
Behavior trees



(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.

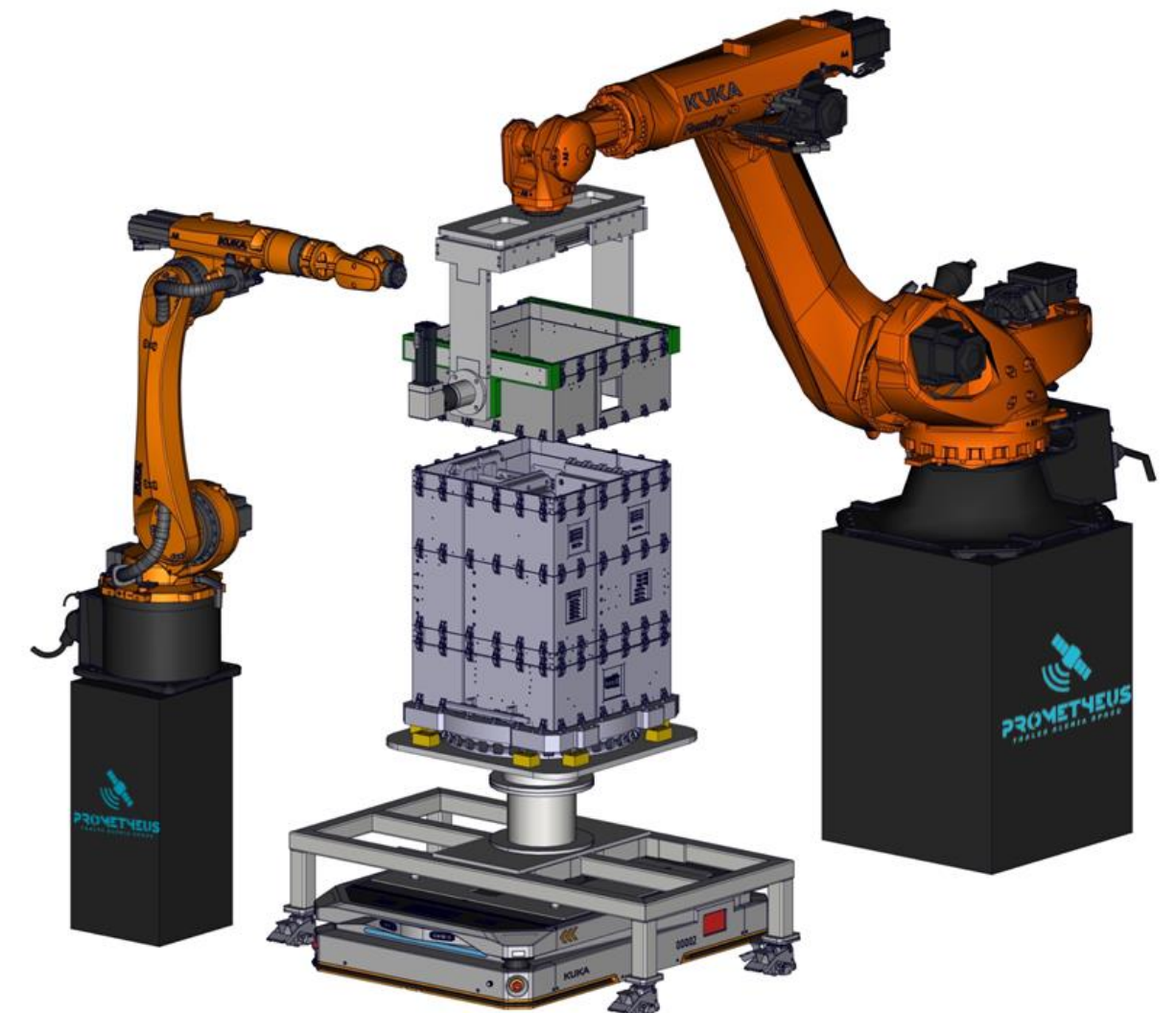


Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	→	If all children succeed	If one child fails	If one child returns Running
Parallel	⇒	If $\geq M$ children succeed	If $> N - M$ children fail	else
Action	text	Upon completion	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:

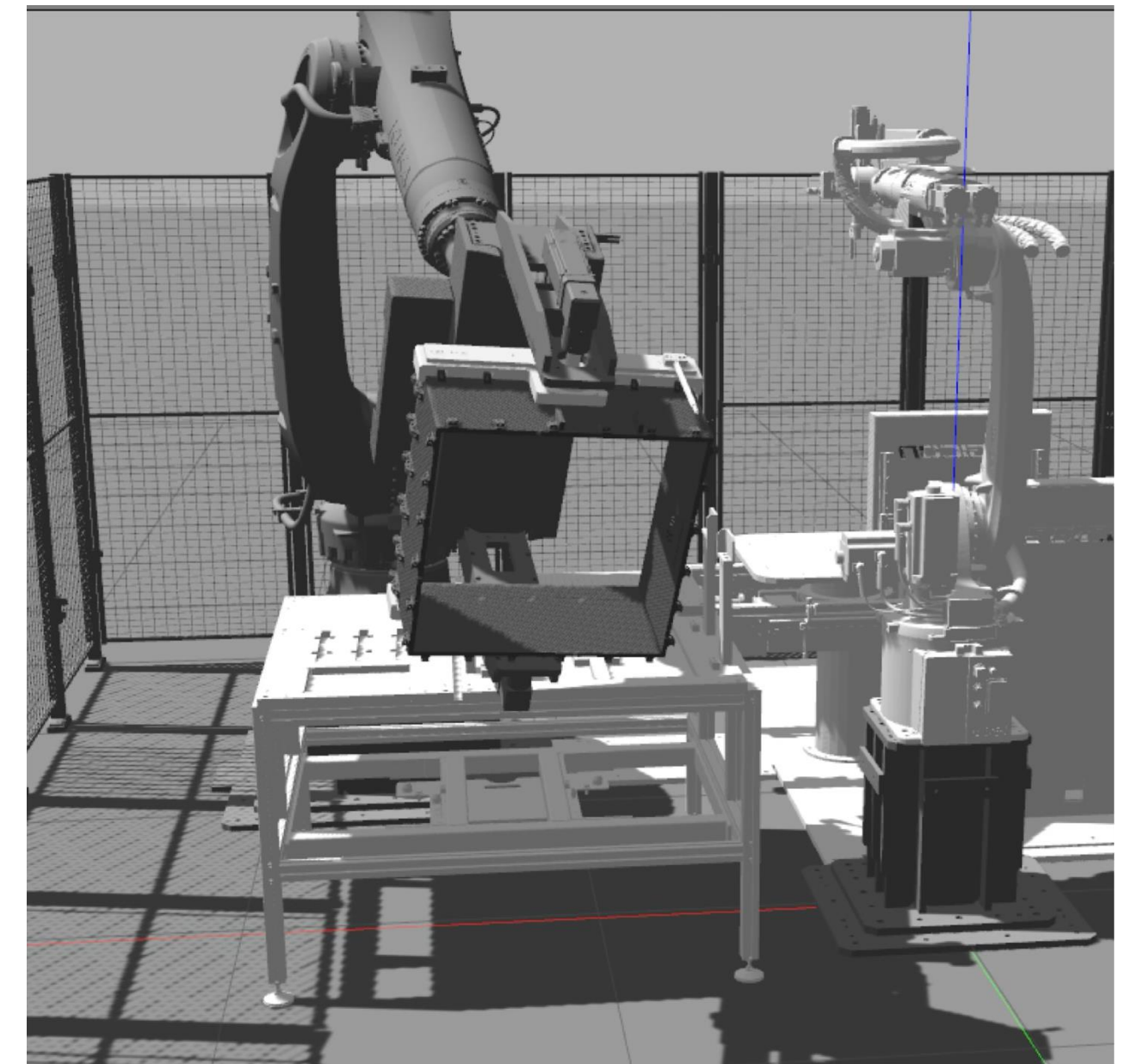
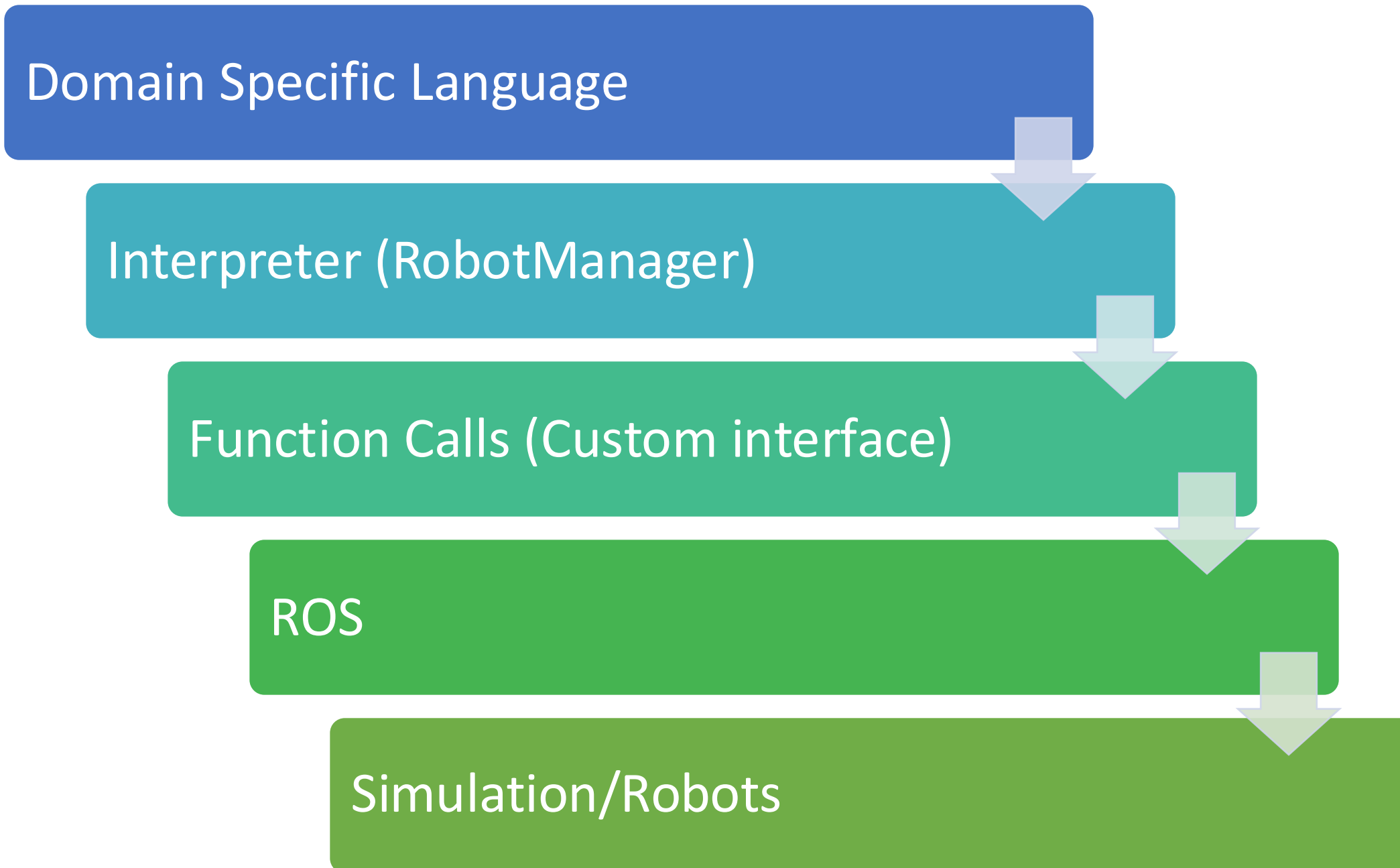
- AOCS Tray, DHC Tray, Components (CMG, MTQ, MAG, screws)



Mission Tasks:

- Moving, Picking, Placing, Screwing, Assembling

Workflow



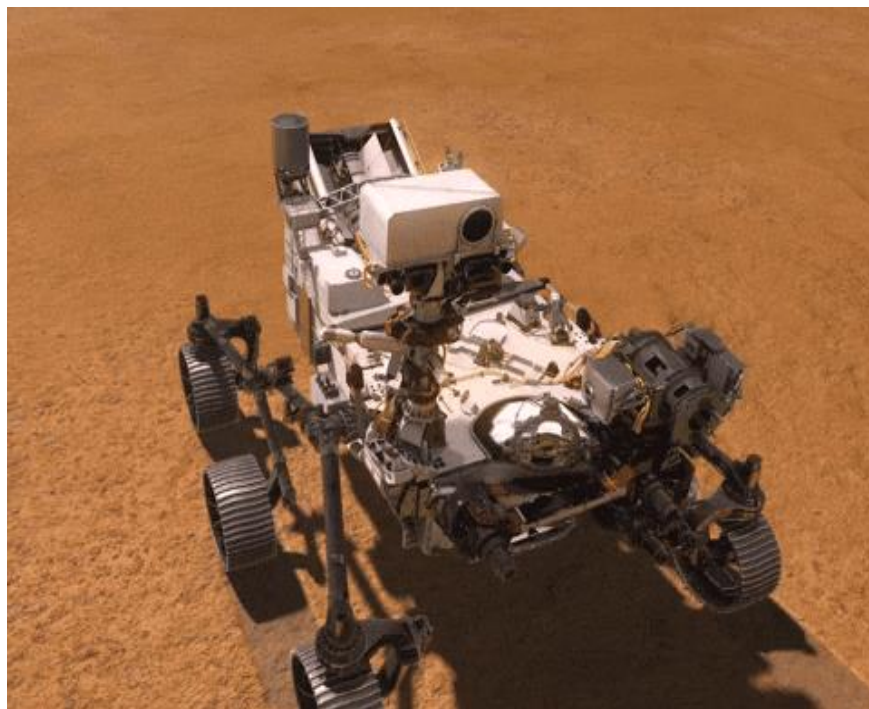
Video

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

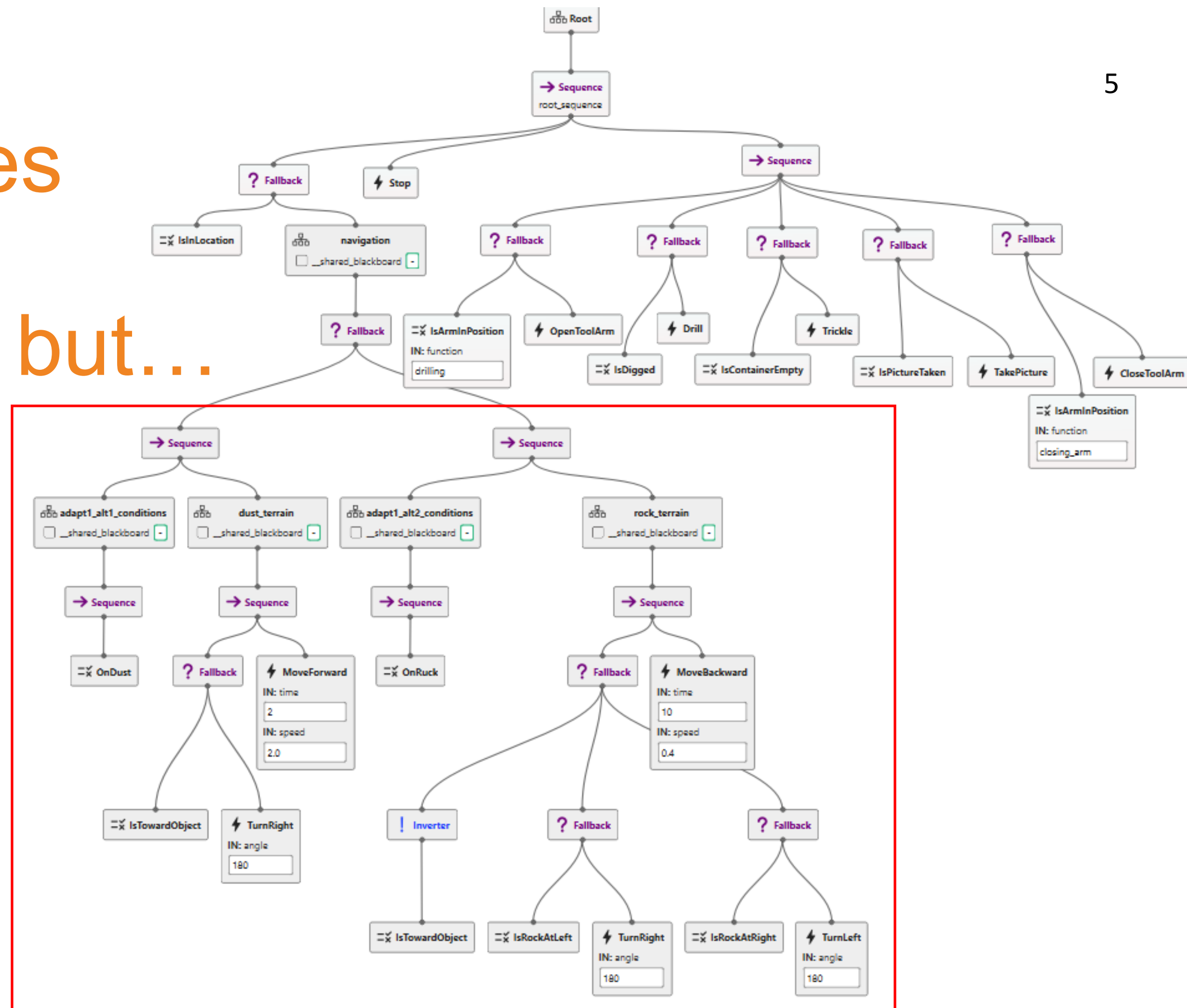
Effects of events/conditions may be unknown at runtime

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

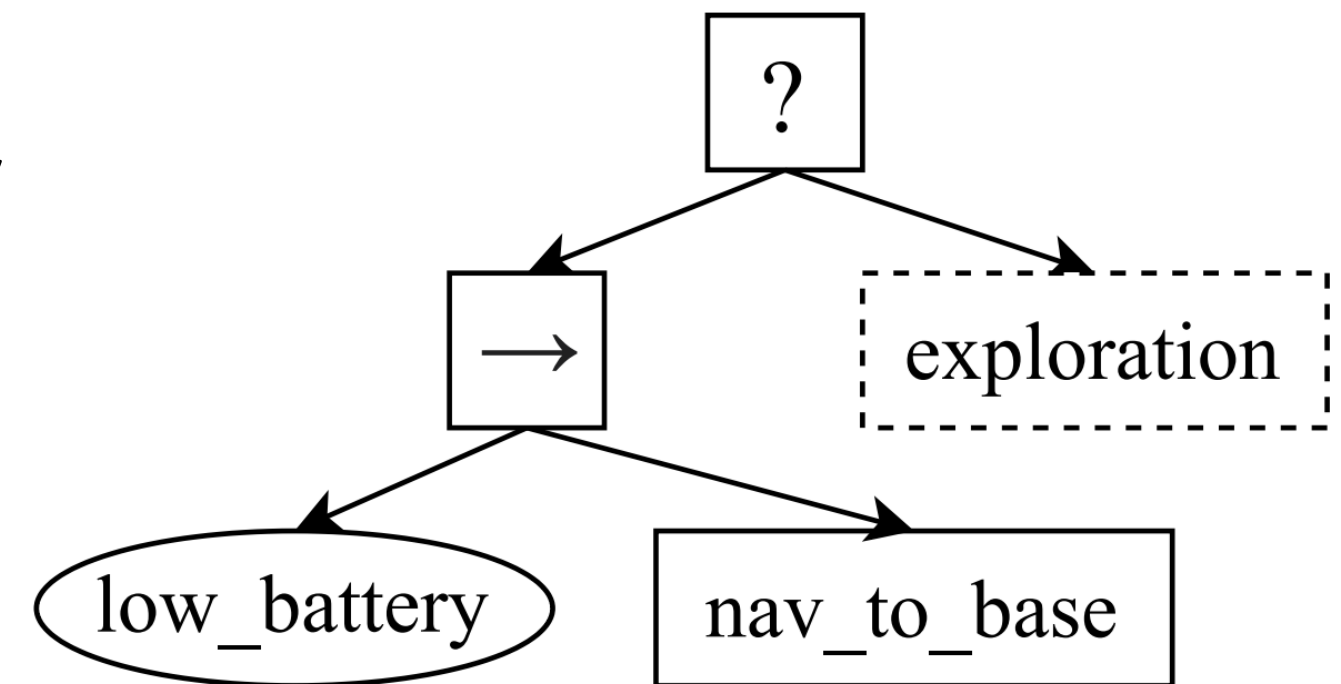
Increase modularity and flexibility

Allow behaviors addition/update

What's missing?

Specification of the conditions for alternatives

Runtime support



Adaptable BT



*What about the unknown
unknown?*

Can LLMs help on that?

Domain Specific Languages

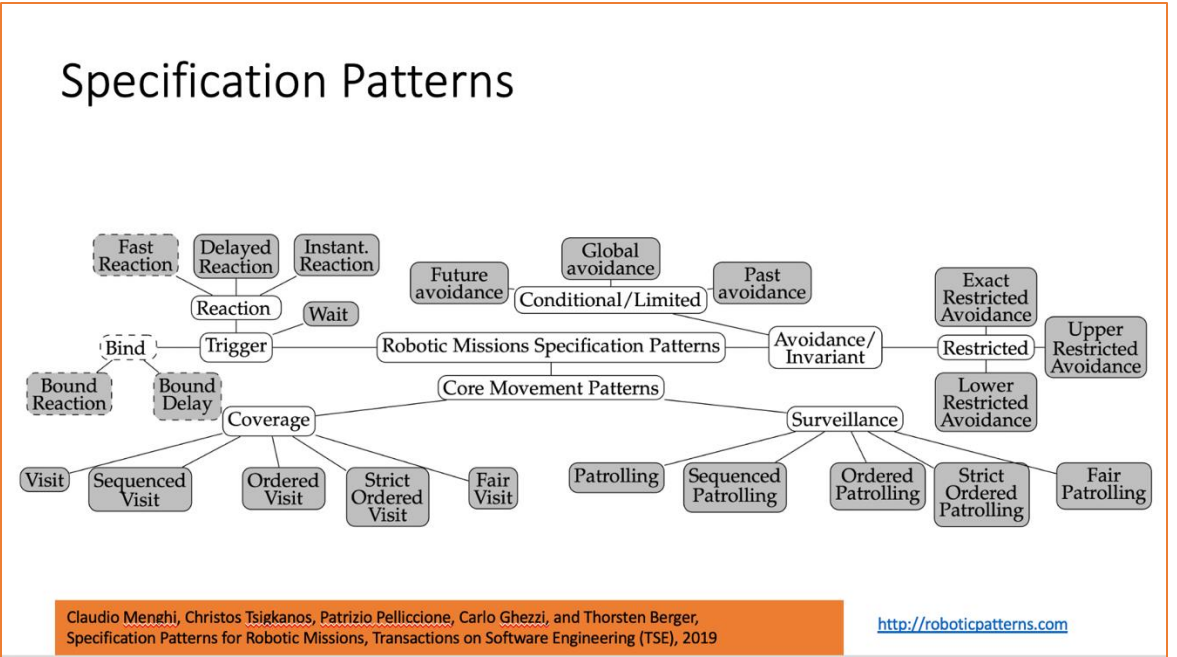
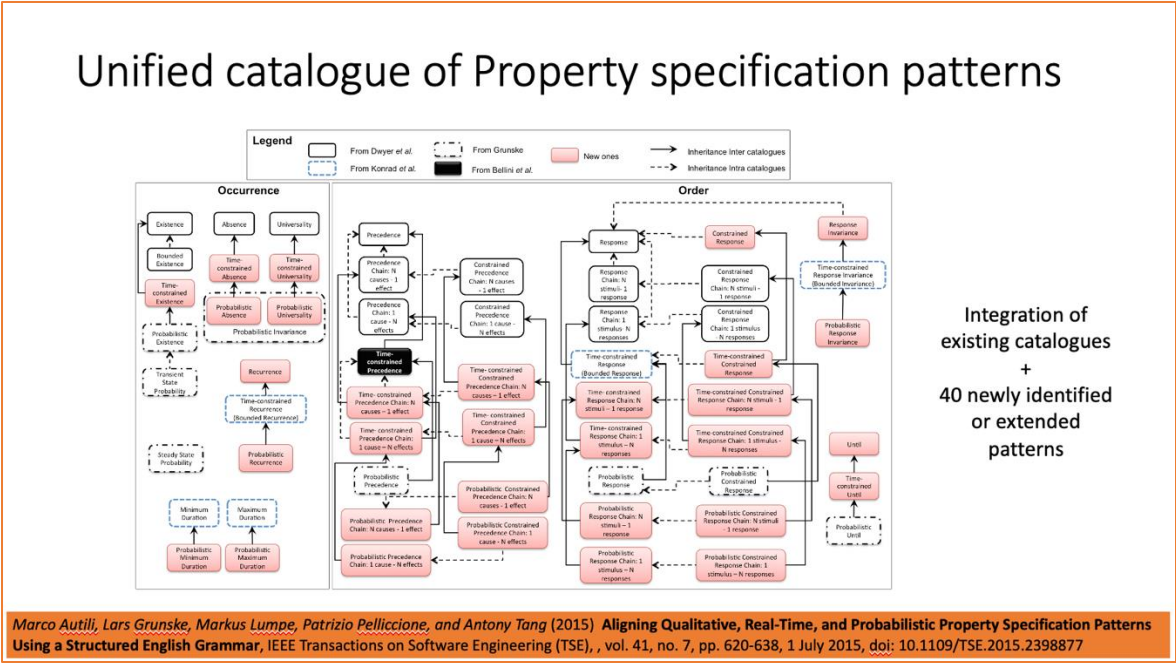
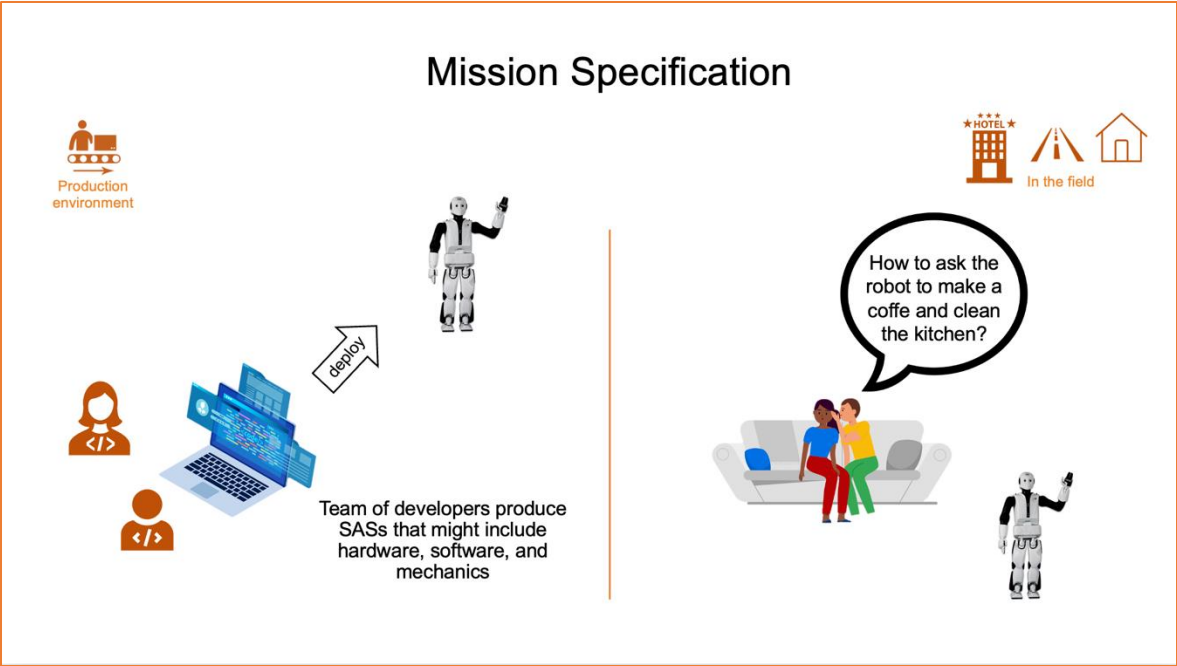
Are the patterns domain specific?

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



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)

The image shows a robotic arm with a gripper, positioned over a manufacturing setup. The arm is orange and black, and the setup includes a metal frame and various components.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiani domani | Fondazione Rome Technopole



GRAN SASSO SCIENCE INSTITUTE



Patrizio Pelliccione

patrizio.pelliccione@gssi.it

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



www.gssi.it

