

Self-Adaptation in Robotics

Ilias Gerostathopoulos

Assistant Professor
Software and Sustainability Group
Computer Science Department
Vrije Universiteit Amsterdam
i.g.gerostathopoulos@vu.nl



**2nd ACM SIGSOFT Summer School for Software
Engineering in Robotics, July 2th 2025**

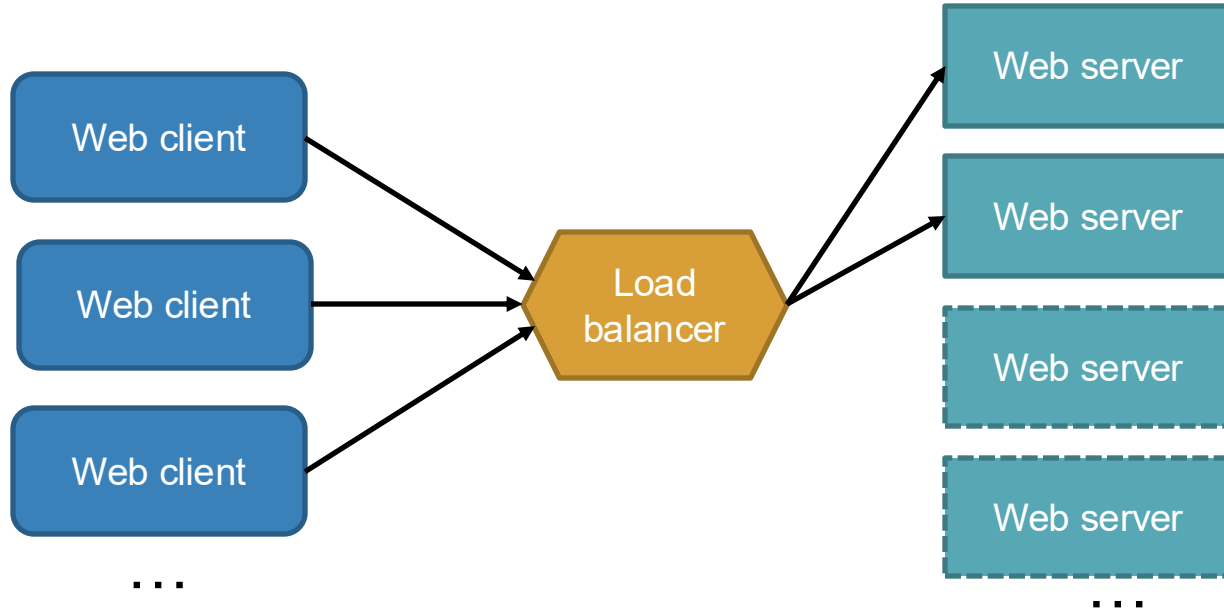
Plan for this lecture

- ▣ Self-adaptive systems (SAS): **Why & What**
 - Preliminaries & Definitions
- ▣ Self-adaptation in robotics: **Two example systems**
 - A biased sample
- ▣ **Architecture-based** self-adaptation
 - A mapping of existing approaches
- ▣ An approach for architecture-based self-adaption
 - Task and architecture **co-adaptation**

Self-adaptive systems (SAS): **Why & What**

Preliminaries

Example #1: Web application



Requests can include optional content (e.g. advertisements)

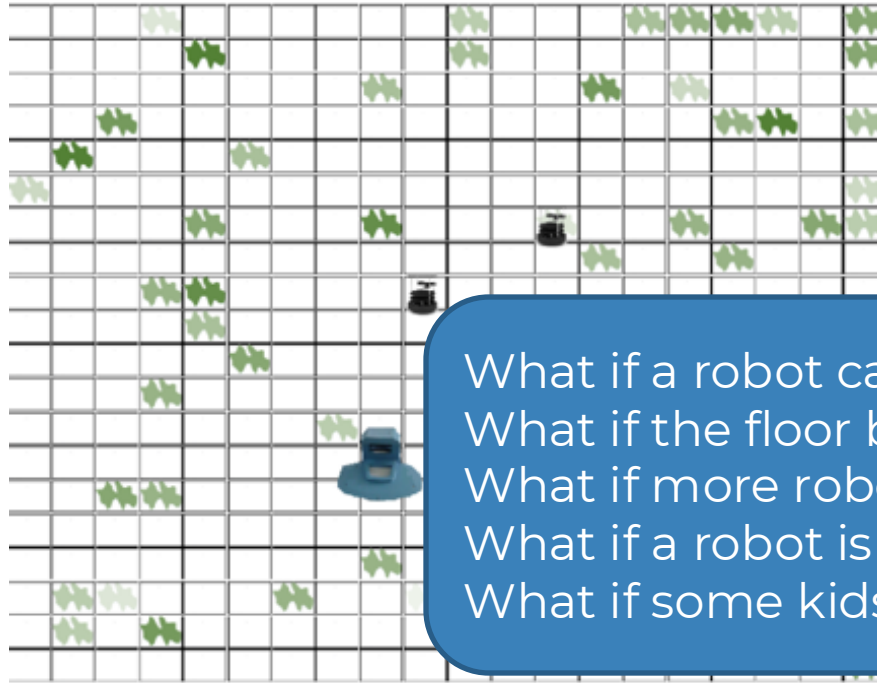
Number of web servers can be increased or decreased (*elastic*)

- ↓ Infrastructure cost
- ↑ Revenue by serving ads
- Latency within bounds

Number and type of requests may fluctuate at runtime (Slashdot-effect)

The actual number and type of requests are uncertainties only known during operation

Example #2: Cleaning robots



What if a robot cannot locate itself anymore?
What if the floor becomes too wet and slippery?
What if more robots join the group?
What if a robot is out of power?
What if some kids start playing with the robot?



Robot



Docking
station



Dirtiness

Why do we need self-adaptation?

Modern systems (incl. robotics!) are subject to uncertainties in

- Their environment (e.g. resources)
- Their users (e.g. number of requests)
- Their internal functioning (e.g. software faults)
- Their goals (e.g. different prioritization of tasks)

**Uncertainties
need to be
handled
during
operation**

Business continuity is essential

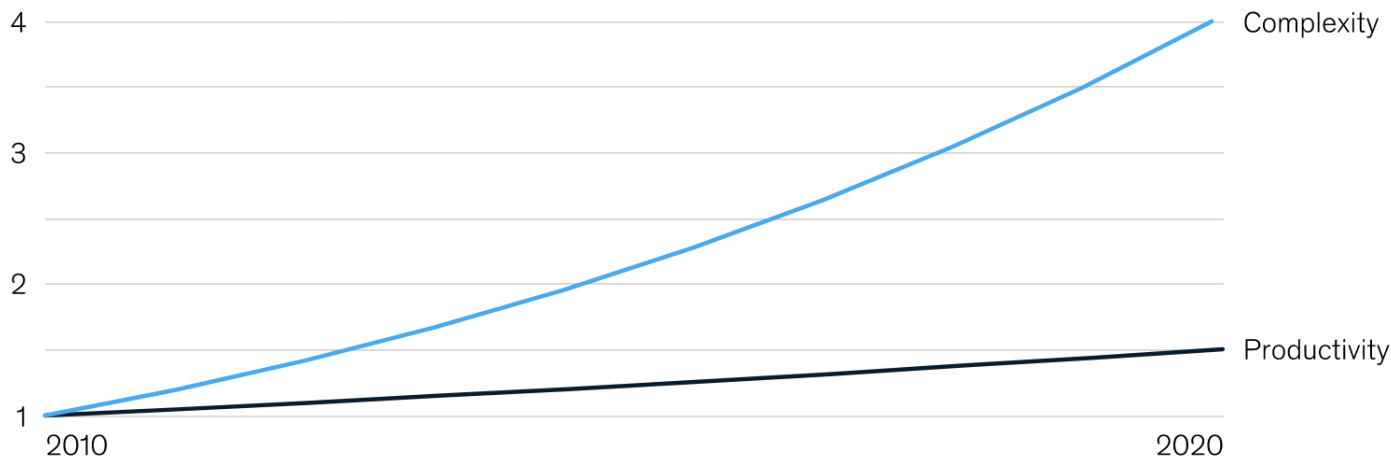
Business continuity in space?

- Stage 1: “Resilient System”
 - *System performs resource management and health management functions. Executes “tactical” activity plans provided by operations team. Uses and adapts models of internal state. Control via closed-loop commanding. Adapts detailed plan to address minor anomalies.*
- Stage 2: “Independent System”
 - *System generates tactical activity plan based on science directives (“strategic plan”) provided by science team. Uses and adapts models of internal state and environment. Possible to reduce size of mission operations team.*
- Stage 3: “Self-Directed System”
 - *System develops science strategic plan and tactical plans based on high-level objectives. Responds to novelty by adjusting plans within context of objectives. Possible to reduce size of science operations team.*

Self-adaptation can bridge the gap

Software complexity is increasing more quickly than productivity.

Relative growth of software complexity and productivity over time, indexed for automotive features



The main idea behind self-adaptation

The system itself (instead of its operators) collects data about its state, environment, and goals **at runtime**, resolves any uncertainties, and adapts to satisfy its goals

Two principles on self-adaptation

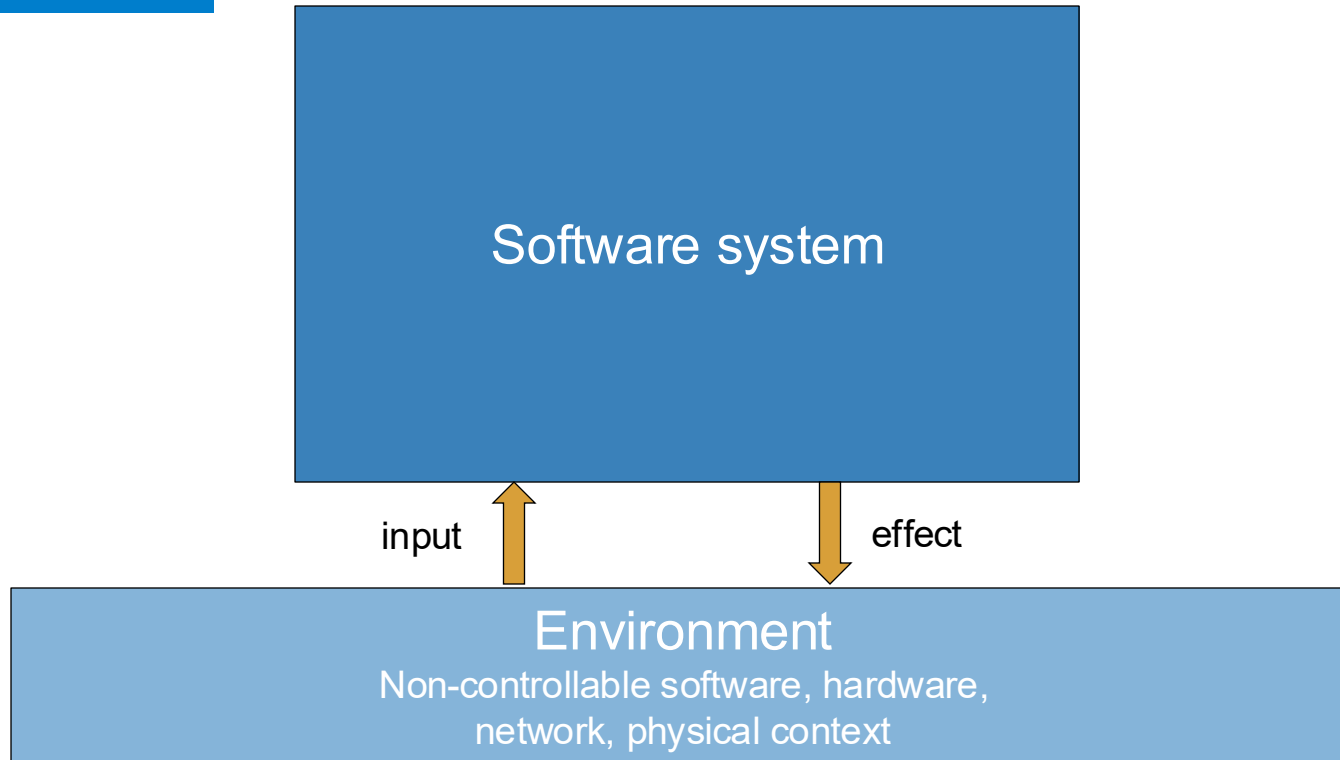
External principle:

- A self-adaptive system is a system that **can handle uncertainty** in its environment, itself and its goals **autonomously** (or with minimal human interference)

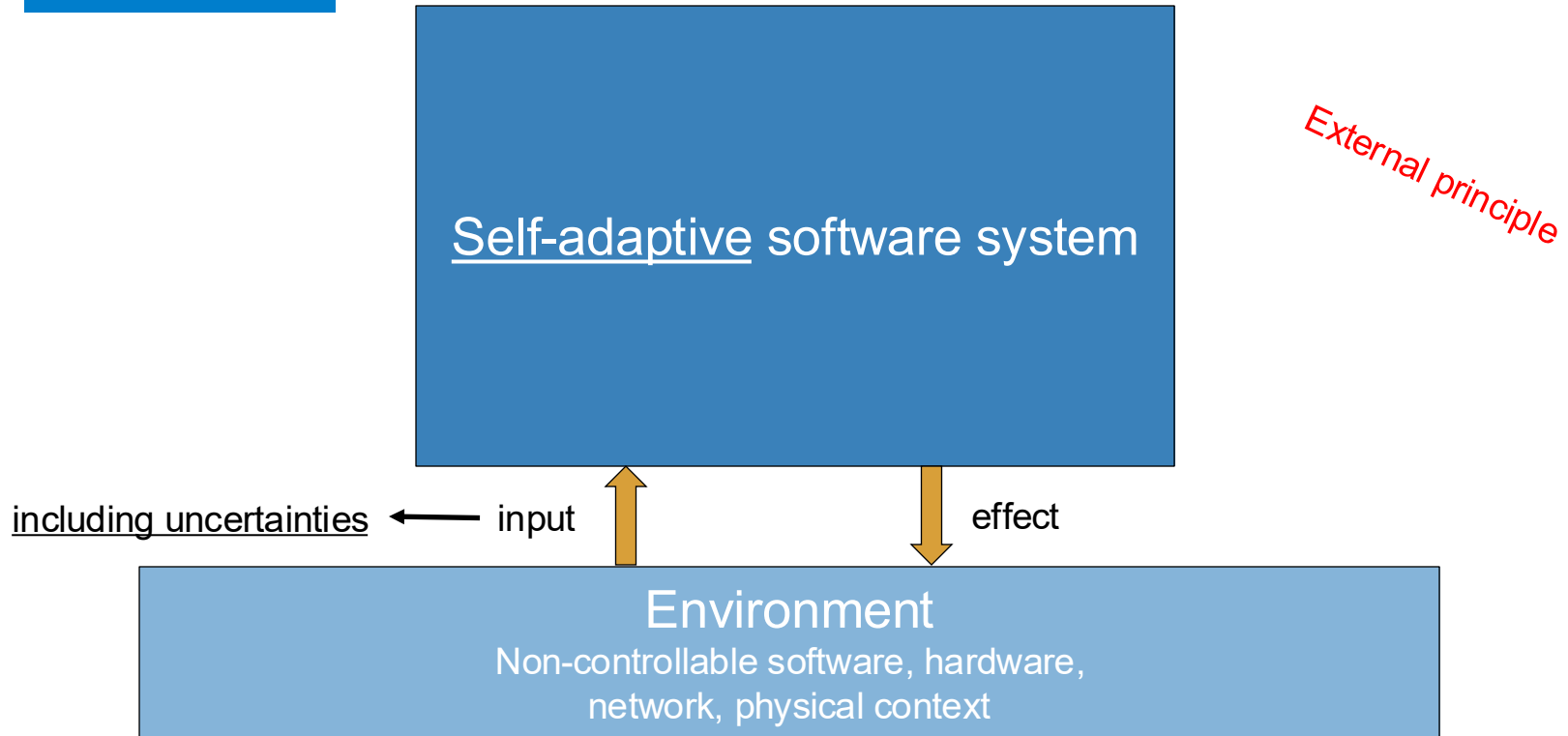
Internal principle

- A self-adaptive system comprises **two distinct parts**: the first part interacts with the environment and has **domain concerns**; the second part interacts with the first part and has **adaptation concerns**, i.e., (usually conflicting) concerns about the domain concerns.

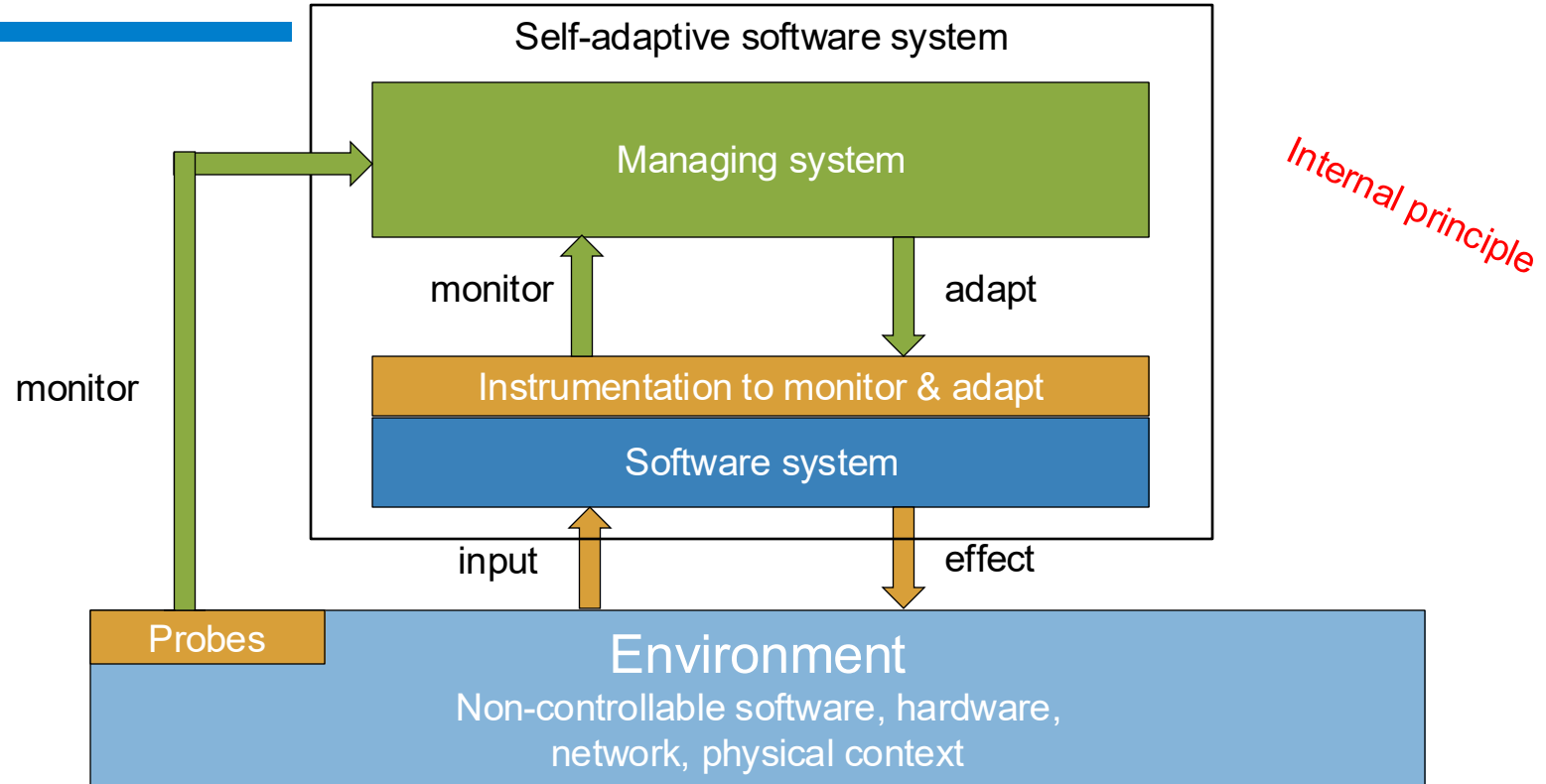
Conceptual model of SAS



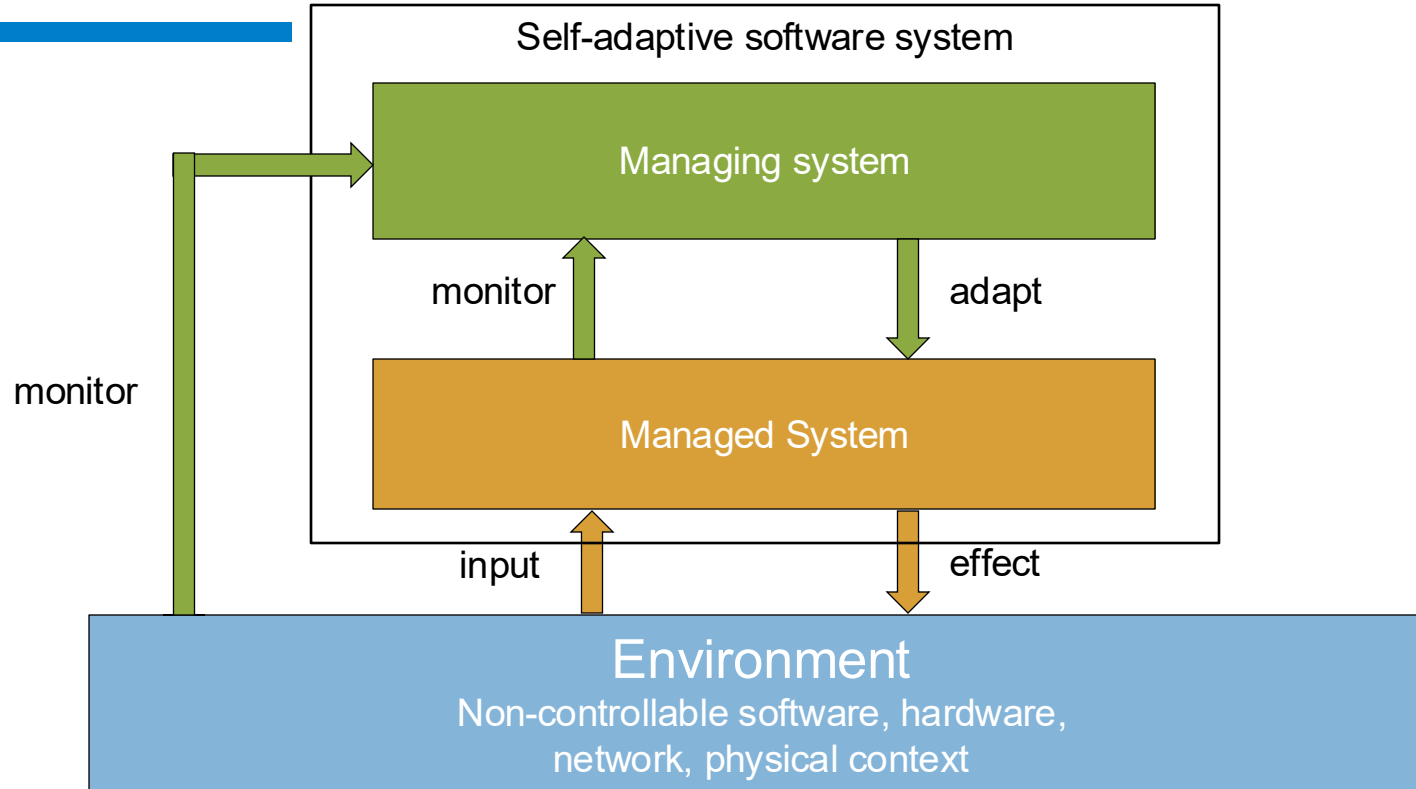
Conceptual model of SAS



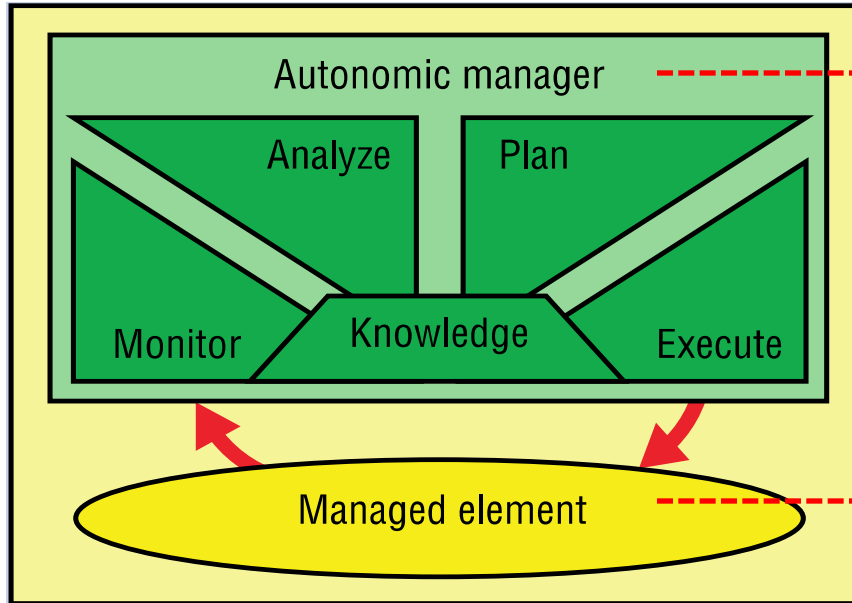
Conceptual model of SAS



Conceptual model of SAS



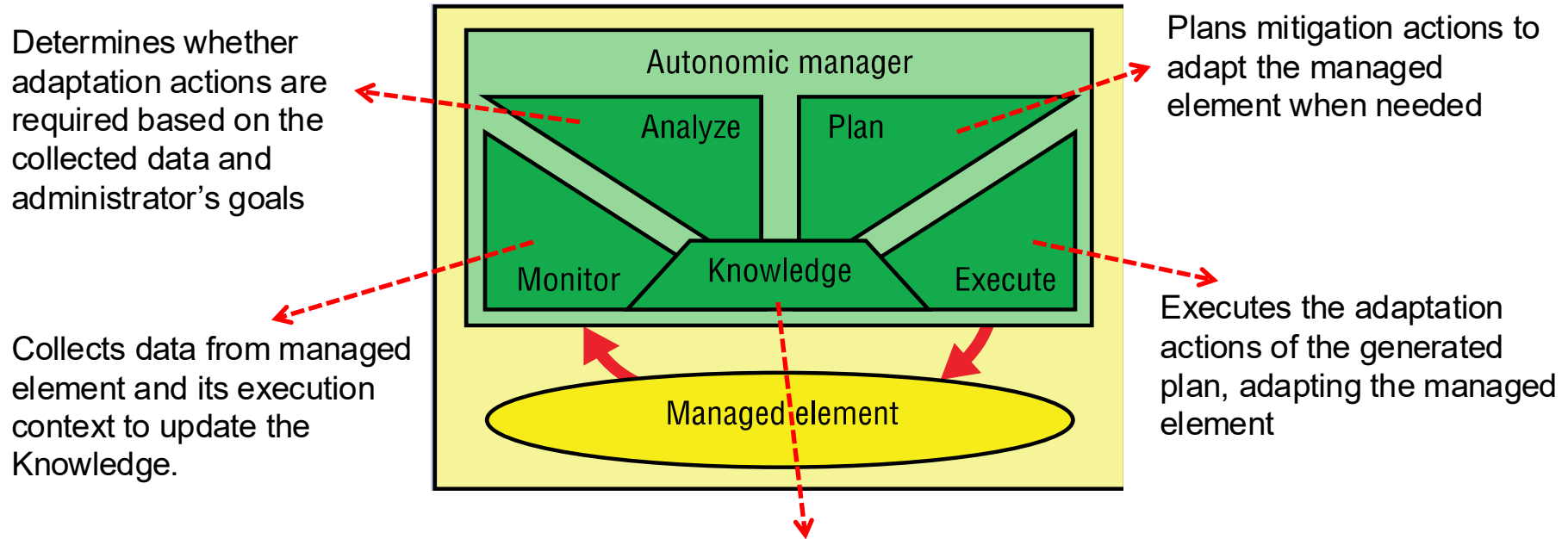
Autonomic manager – MAPE-K reference model



Relieves administrators of responsibility to directly manage the managed element

Element to be monitored and controlled to realize administrator's goals

Autonomic manager – MAPE-K reference model



Self-* properties

General Level

Self-Adaptiveness

Major Level

Self-Configuring

Self-Healing

Self-Optimizing

Self-Protecting

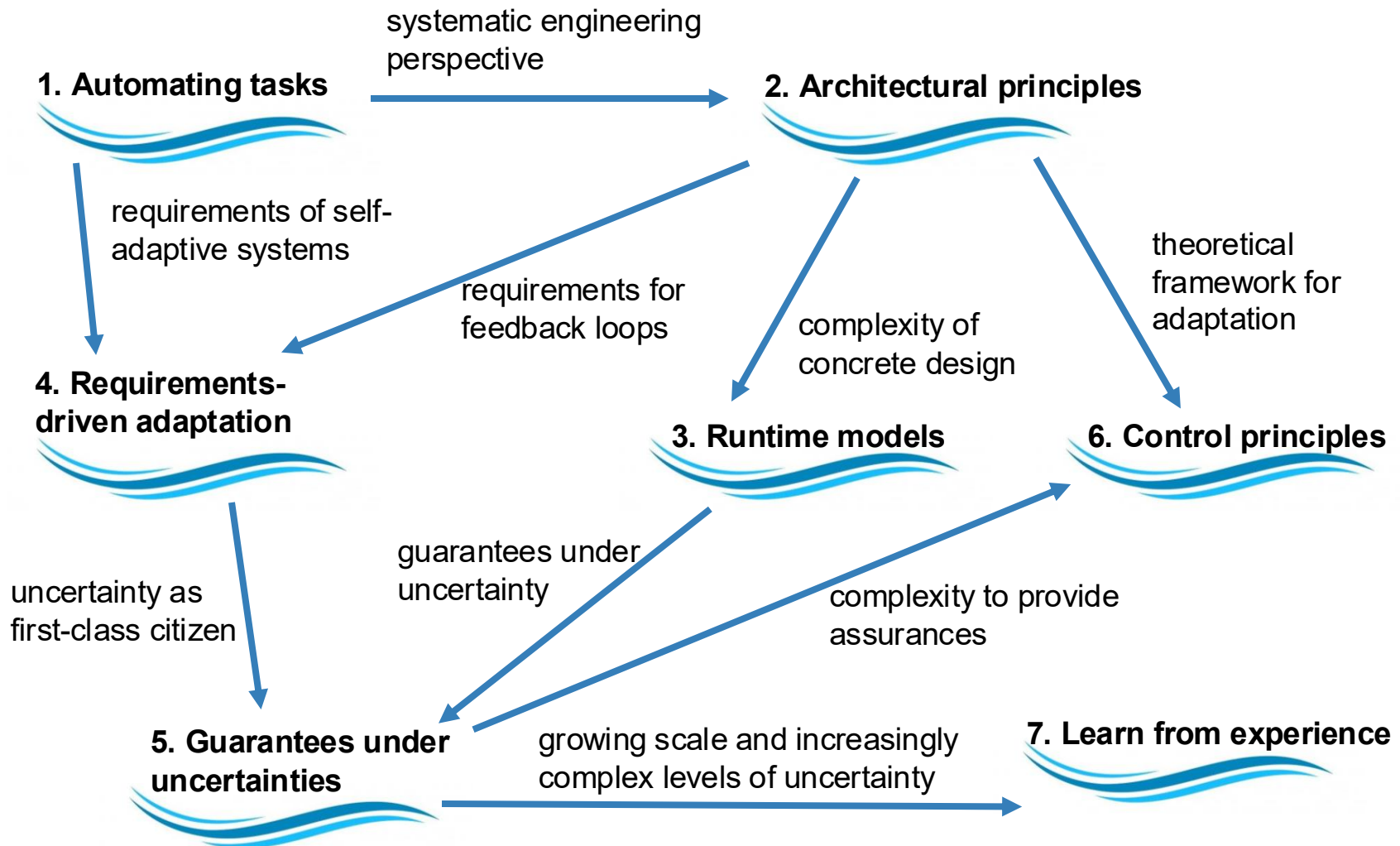
Primitive Level

Self-Awareness

Context-Awareness

The seven waves

- ▣ Research in self-adaptation has been categorized in seven waves by Danny Weyns
- ▣ Focus on how self-adaptive systems are engineered
- ▣ Highlight **research trends** and their influences
- ▣ Contribute **complementary layers** of knowledge

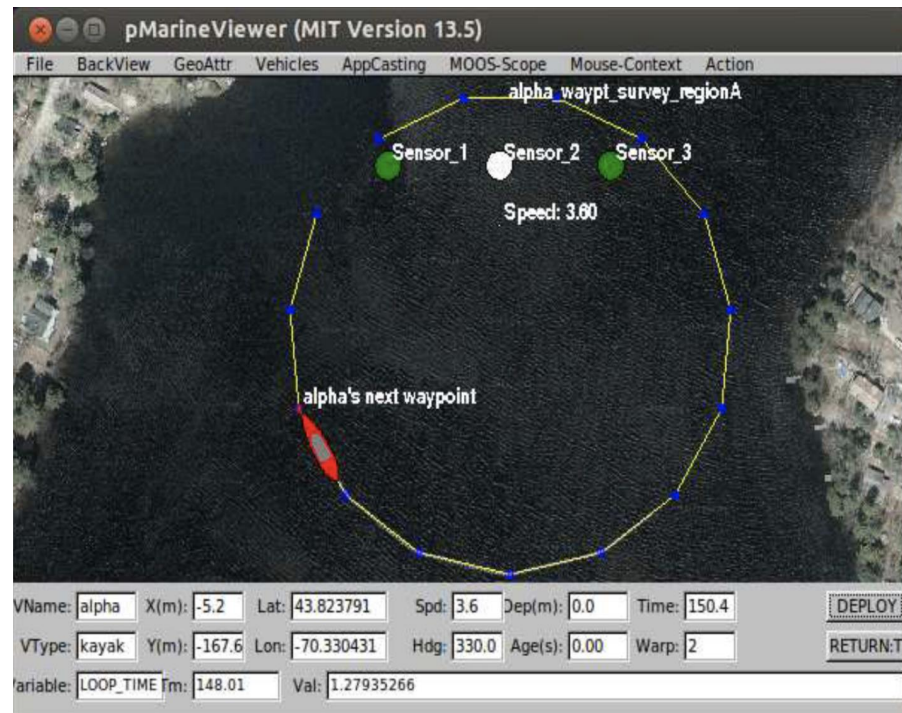


Self-adaptation in robotics: **Two example systems**

A biased sample

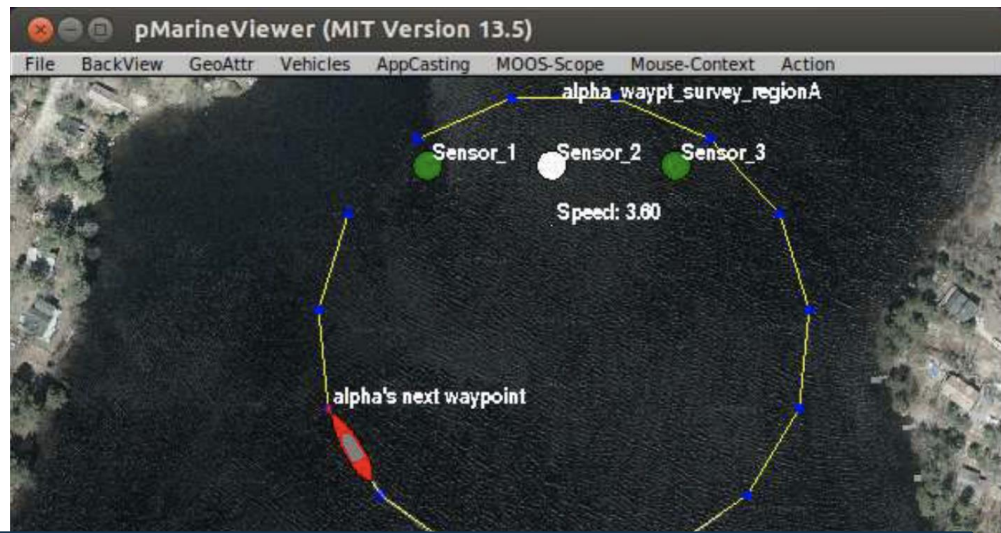
UNDERSEA

- Unmanned Underwater Vehicle (UUV) on an environmental surveillance mission
 - Contains sensors (water current, salinity, temperature)
 - Each sensor with rate and reliability



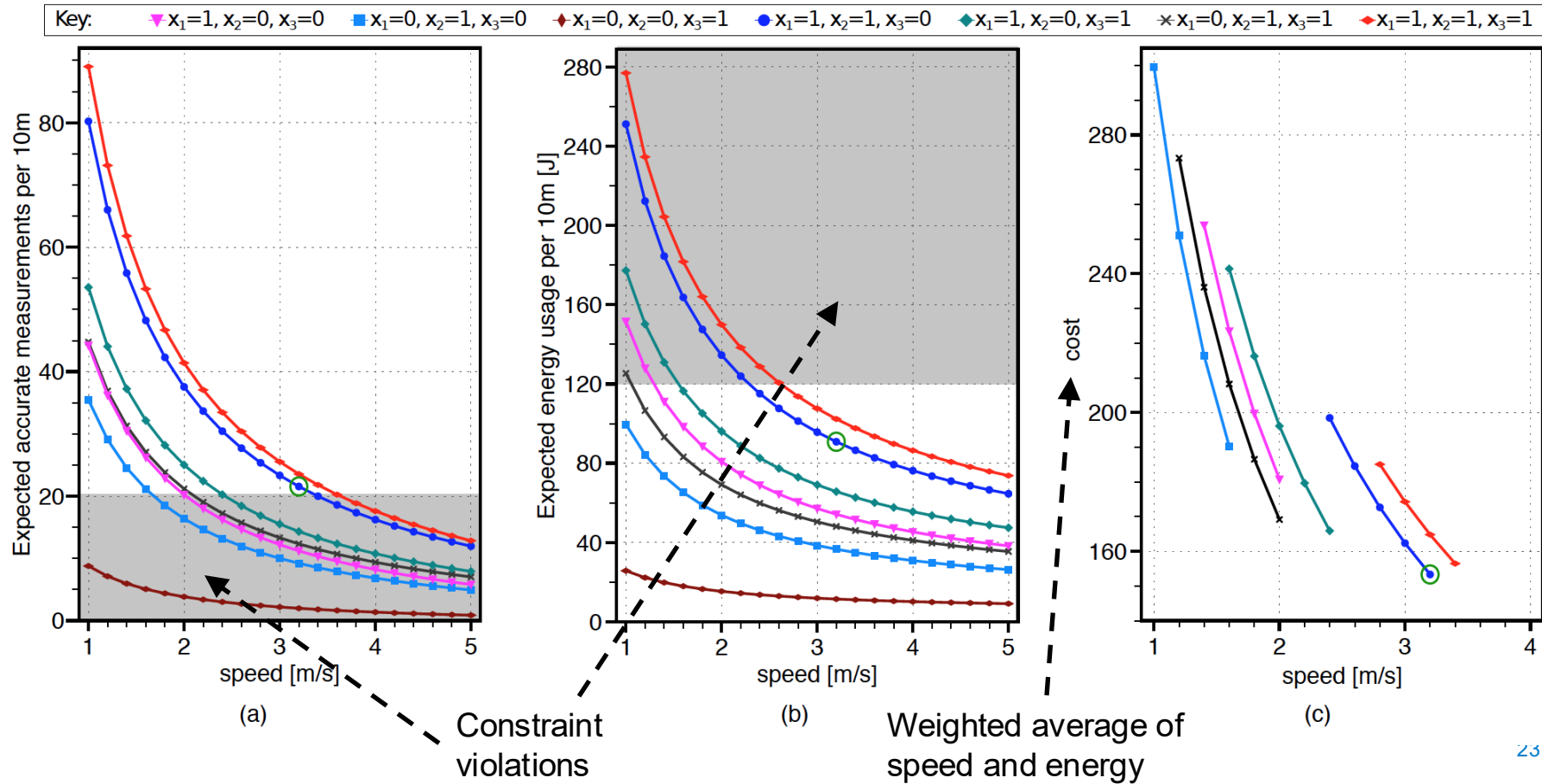
UNDERSEA: possible runtime changes

- ▣ Increasing/decreasing speed of the UUV
- ▣ Turning sensors on/off (assuming they measure the same thing, e.g., temperature)



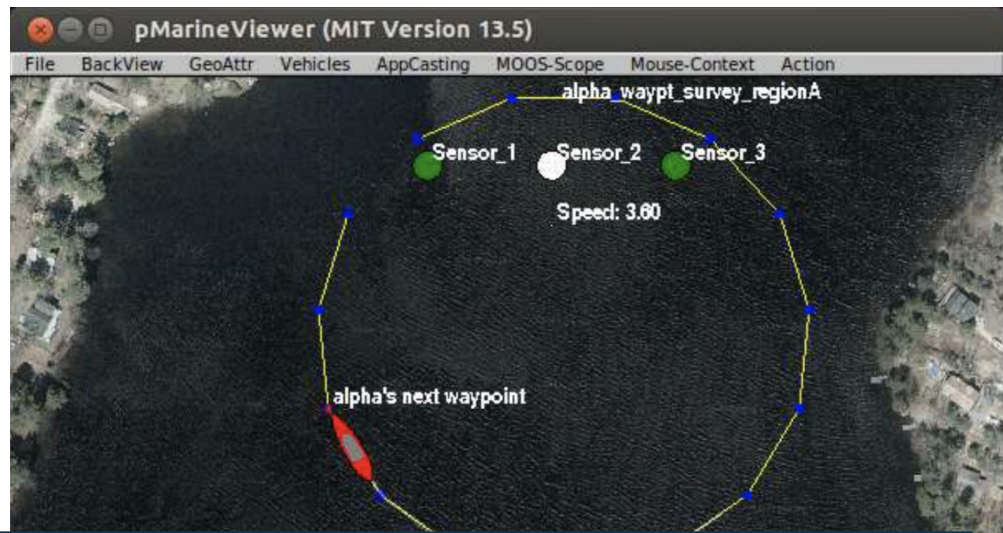
Given constraints on number of “good” measurements & energy per X meters, minimize energy, maximize speed

UNDERSEA: let's pick a configuration



UNDERSEA: what can go wrong?

- ❑ Sensor degradation leading to lower sensing rates
- ❑ Sensor failure (sensor cannot be used anymore)
- ❑ Change of constraint or optimization goal



Now, the configuration “optimization problem” needs to be solved at runtime!

UNDERSEA: runtime reasoning (PRISM, CTMP)

```
module sensor1
```

```
  // system states
```

```
  stateS1 : [0..6] init 0; // 0:start - 1:on - 2:read - 3:succ - 4: fail - 5:done - 6:off
```

```
  [switchS1]          (stateS1=0) & (sensor1Enabled) -> 1000.0 : (stateS1'=1);
```

```
  [switchS1]          (stateS1=0) & (!sensor1Enabled) -> 1000.0 : (stateS1'=6);
```

```
  [readS1]            (stateS1=1) -> r1 : (stateS1'=2);
```

```
  [succReadS1]        (stateS1=2) -> p1 : (stateS1'=3);
```

```
  []                  (stateS1=2) -> (100.0 - p1): (stateS1'=4);
```

```
  []                  (stateS1=3) -> 1000.0 : (stateS1'=5);
```

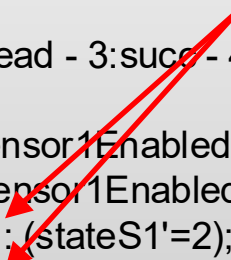
```
  []                  (stateS1=4) -> 1000.0 : (stateS1'=5);
```

```
  []                  (stateS1=5) -> 1000.0 : (stateS1'=1);
```

```
  []                  (stateS1=6) -> 1000.0 : (stateS1'=6);
```

```
endmodule
```

Model parameters (their values
measurable at runtime)



UNDERSEA: runtime reasoning (PRISM, CTMP)

```
module sensor1
```

```
  // system states
```

```
  stateS1 : [0..6] init 0; // 0:start - 1:on - 2:read - 3:succ - 4: fail - 5:done - 6:off
```

```
  [switchS1]
```

```
  [switchS1]
```

```
  [readS1]
```

```
  [succReadS1]
```

```
  []
```

```
  []
```

```
  []
```

```
  []
```

```
  []
```

```
endmodule
```

Model parameters (their values
measurable at runtime)

```
rewards "energy"
```

```
  [readS1] true : 3;
```

```
  [readS2] true : 2.4;
```

```
  [readS3] true : 2.1;
```

```
  [switchS1] true : sensor1SwitchCost;
```

```
  [switchS2] true : sensor2SwitchCost;
```

```
  [switchS3] true : sensor3SwitchCost;
```

```
endrewards
```

UNDERSEA: runtime reasoning (PRISM, CTMP)

```
module sensor1
```

```
  // system states
```

```
  stateS1 : [0..6] init 0; // 0:start - 1:on - 2:read - 3:succ - 4: fail - 5:done - 6:off
```

```
  [switchS1]
```

```
  [switchS1]
```

```
  [readS1]
```

```
  [succReadS1]
```

```
  []
```

```
  []
```

```
  []
```

```
  []
```

```
  []
```

```
endmodule
```

Model parameters (their values
measurable at runtime)

```
rewards "energy"
```

```
  [readS1] true : 3;
```

```
  [readS2] true : 2.4;
```

```
  [readS3] true : 2.1;
```

```
  [switchS1] true : sensor1SwitchCo
```

```
  [switchS2] true : sensor2SwitchCo
```

```
  [switchS3] true : sensor3SwitchCo
```

```
endrewards
```

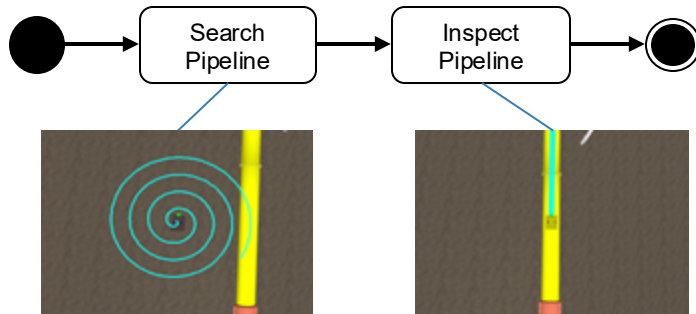
$R\{\text{"measurement"}\}=? [C \leq 10/s]$

$R\{\text{"energy"}\}=? [C \leq 10/s]$

SUAVE

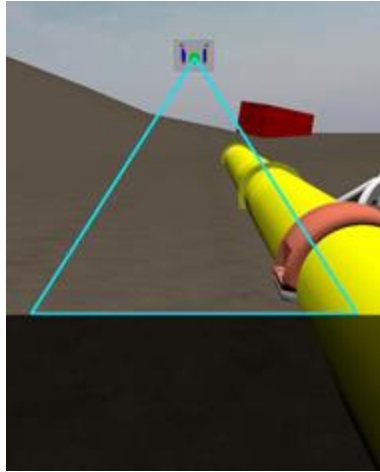
Self-Adaptive Underwater Autonomous Vehicles Exemplar

Scenario: pipeline inspection for a single robot

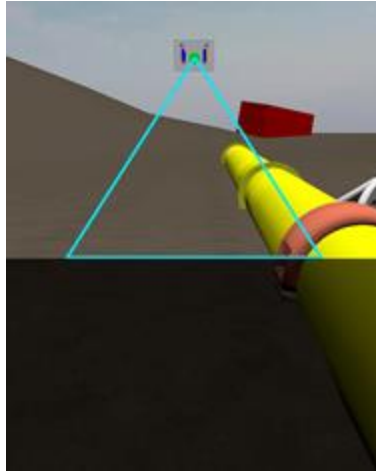


SUAVE: uncertainties + feedback loops

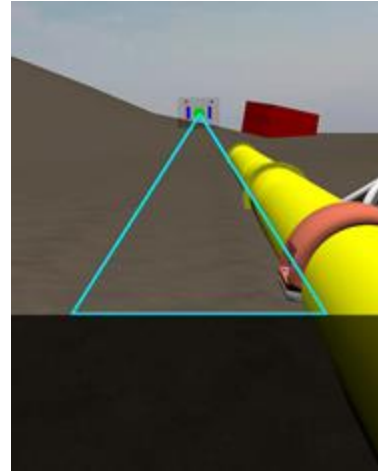
1. Water visibility may change during the mission



High water visibility and high altitude



Low water visibility and high altitude



Low water visibility and low altitude

SUAVE: uncertainties + feedback loops

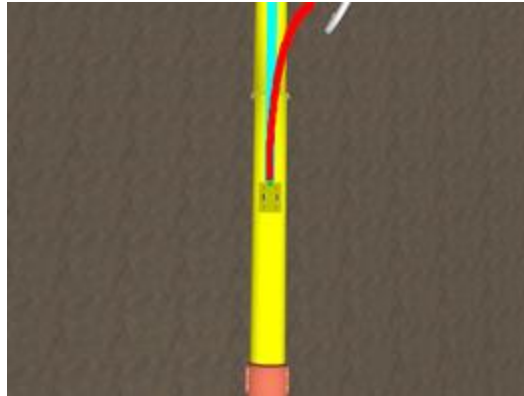
1. Water visibility may change during the mission
 - a. Can this influence the mission?
 - Yes, the effectiveness of searching may be decreased if the visibility is low
 - b. What would we like the robot to do if it would be able to resolve this uncertainty at runtime?
 - Go higher when the visibility is high (so that its field of view is larger)
 - c. Can the robot monitor the visibility at runtime?
 - Yes, with a turbidimeter/nephelometer

SUAVE: uncertainties + feedback loops

1. Water visibility may change during the mission
 - d. Can the robot change its search strategy at runtime?
 - Yes, by selecting a different depth to search (implemented by changing the configuration of the search function or selecting between search functions for different depths)
 - e. Does it pay off to implement the above feedback loop?
 - Our results indicate that the time to find the pipeline with the loop present gets almost half→ less time equals less fuel, more inspection time

SUAVE: uncertainties + feedback loops

2. One of its six thrusters may fail/malfunction



Thruster failure cause the
AUV to deviate from its path

SUAVE: uncertainties + feedback loops

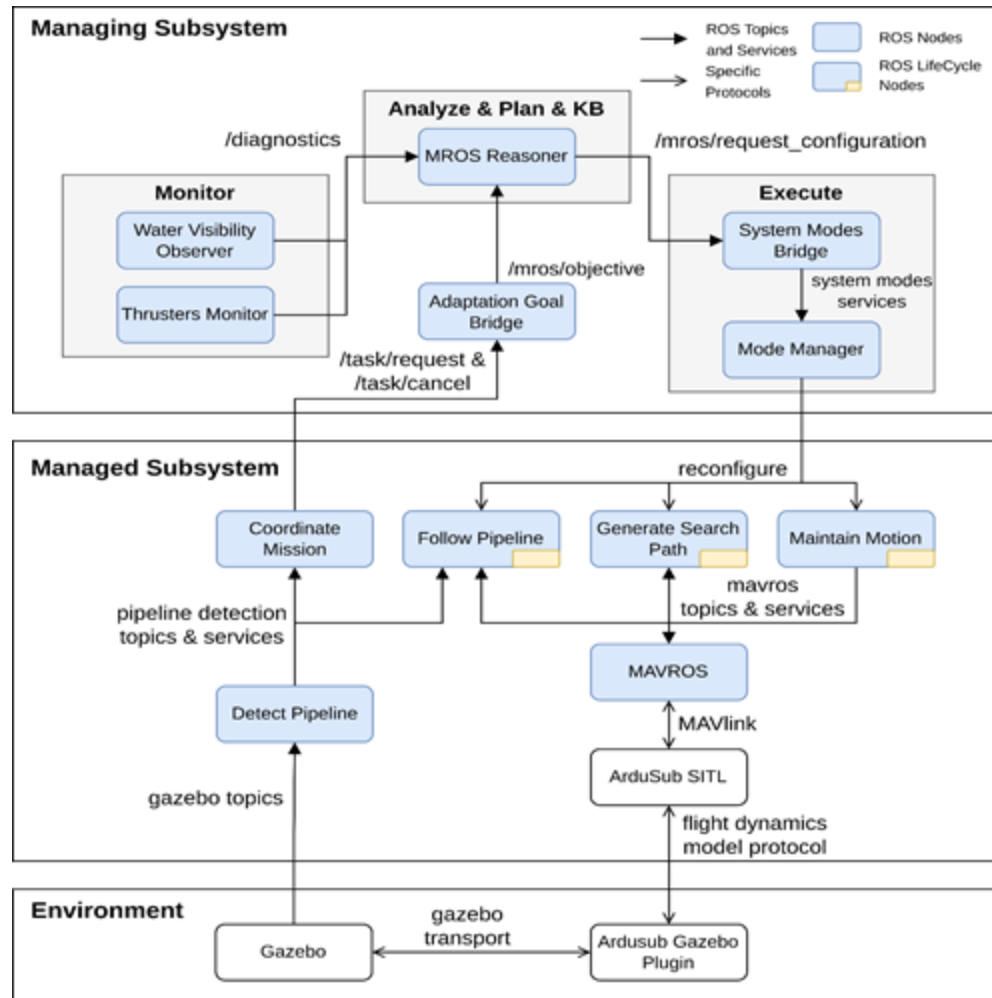
2. One of its six thrusters may fail/malfunction
 - a. Can this influence the mission?
 - Yes, the robot may not be able to follow its autopilot anymore and stray forever and ever
 - b. What would we like the robot to do if it would be able to resolve this uncertainty at runtime?
 - Fix or replace the failing thruster, or use a back-up one, or just use the remaining thrusters
 - c. Can the robot monitor the thruster failure at runtime?
 - Yes, through Finite Impulse Response (FIR) and Principal Component Analysis (PCA)

SUAVE: uncertainties + feedback loops

2. One of its six thrusters may fail/malfunction
 - d. Can the robot deal with failed/malfunctioning thrusters at runtime?
 - d. Yes, by restarting them (assumption!)
 - e. Does it pay off to implement the above feedback loop?
 - Our results indicate that the length of the pipeline inspected is increased by 50% with the loop → more efficient missions

SUAVE: technical architecture

- Essentially, water visibility and thruster states are monitored and ROS2 components are reconfigured
- MROS, System modes



SUAVE is available on Github

Give it a try!



Screenshot of the GitHub repository page for **kas-lab / suave**.

The repository is public and has 536 commits, 9 forks, and 24 stars. It is categorized under **robotics**, **ros**, **ros2**, **self-adaptive-systems**, and **marine-robotics**.

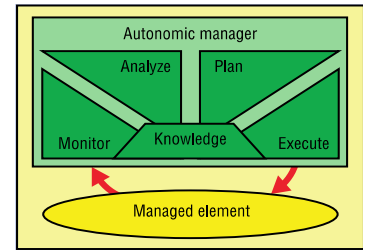
The repository description is: **An Exemplar for Self-Adaptive Underwater Vehicles performing pipeline inspection**. The repository link is kas-lab.github.io/suave/.

The repository structure shows the following files and folders:

- .github/workflows**: add suave_monitor and suave_metrics to CI file (3 weeks ago)
- docker**: fix dockerfile for new suave and ardupilot_plugin versi... (last week)
- docs**: delete files generated by sphinx (3 weeks ago)
- runner**: add support for bt in the runner (13 hours ago)
- suave**: change 127.0.0.0 to 0.0.0.0 in mavros launch (17 hours ago)
- suave_managing**: remove SuaveReasoner (13 hours ago)
- suave_metrics**: get time from msg (13 hours ago)
- suave_missions**: adjust suave_mission launchfile (17 hours ago)
- suave_monitor**: publish battery_level as soon as it charges (last week)
- suave_msgs**: add task srv (last year)

The repository is managed by **Rezenders** and includes a merge pull request #169 from kas-lab/mc_reasoner. The repository is also available on **scite_** with a score of 1.0.

Exercise



- I. Identify some uncertainties in the robotic systems you are working with
- II. For each uncertainty, answer the following:
 - a. Can it influence the mission?
 - b. What would we like the system to do if it would be able to resolve this uncertainty at runtime?
 - c. Can the robot monitor at runtime quantities that can resolve the uncertainty?
 - d. Can the robot change its behavior at runtime to recover or optimize itself?
 - e. Does it pay off to implement the above feedback loop?

Architecture-based self-adaptation

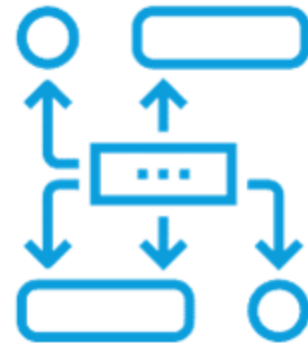
A mapping of existing approaches

What is software architecture?

- ▣ Fundamental structure of a software system
- ▣ Important is the process to arrive to the structures: architectural decisions
- ▣ Related terms:
 - Design patterns (e.g. Gang of Four* patterns)
 - Architectural styles (e.g. MVC)
 - Component models (e.g. OSGI)

Why architecture-based self-adaptation?

- ▣ Separation of concerns
- ▣ Integrated approach
- ▣ Leveraging consolidated efforts
- ▣ Abstraction to manage system change
- ▣ Dealing with system-wide concerns
- ▣ Facilitating scalability



3-Layer model

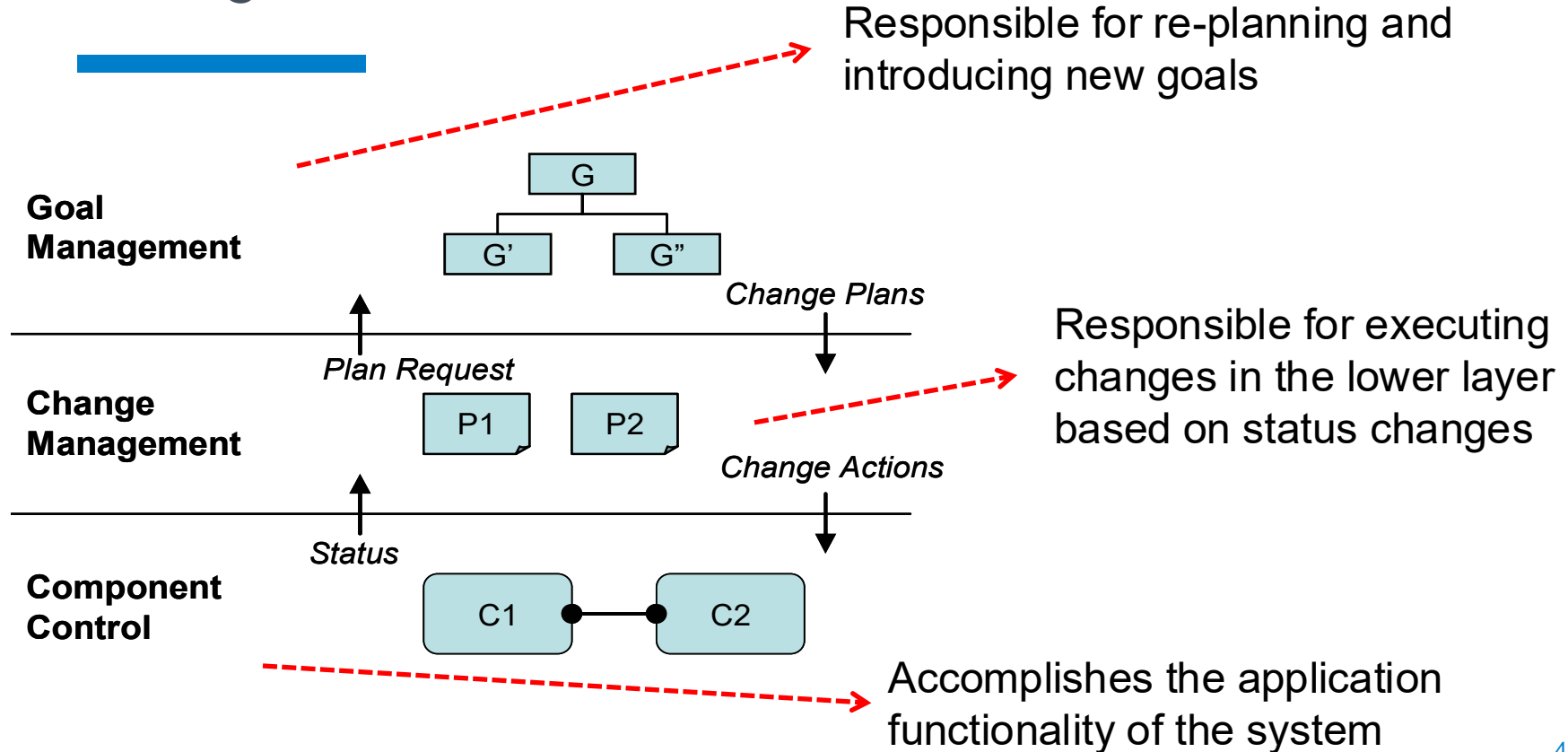
“Components automatically configure their interaction in a way that is compatible with an overall architectural specification and achieves the goals of the system.”

“... the architectural level seems to provide the required level of abstraction and generality to deal with the challenges posed by self-adaption.”

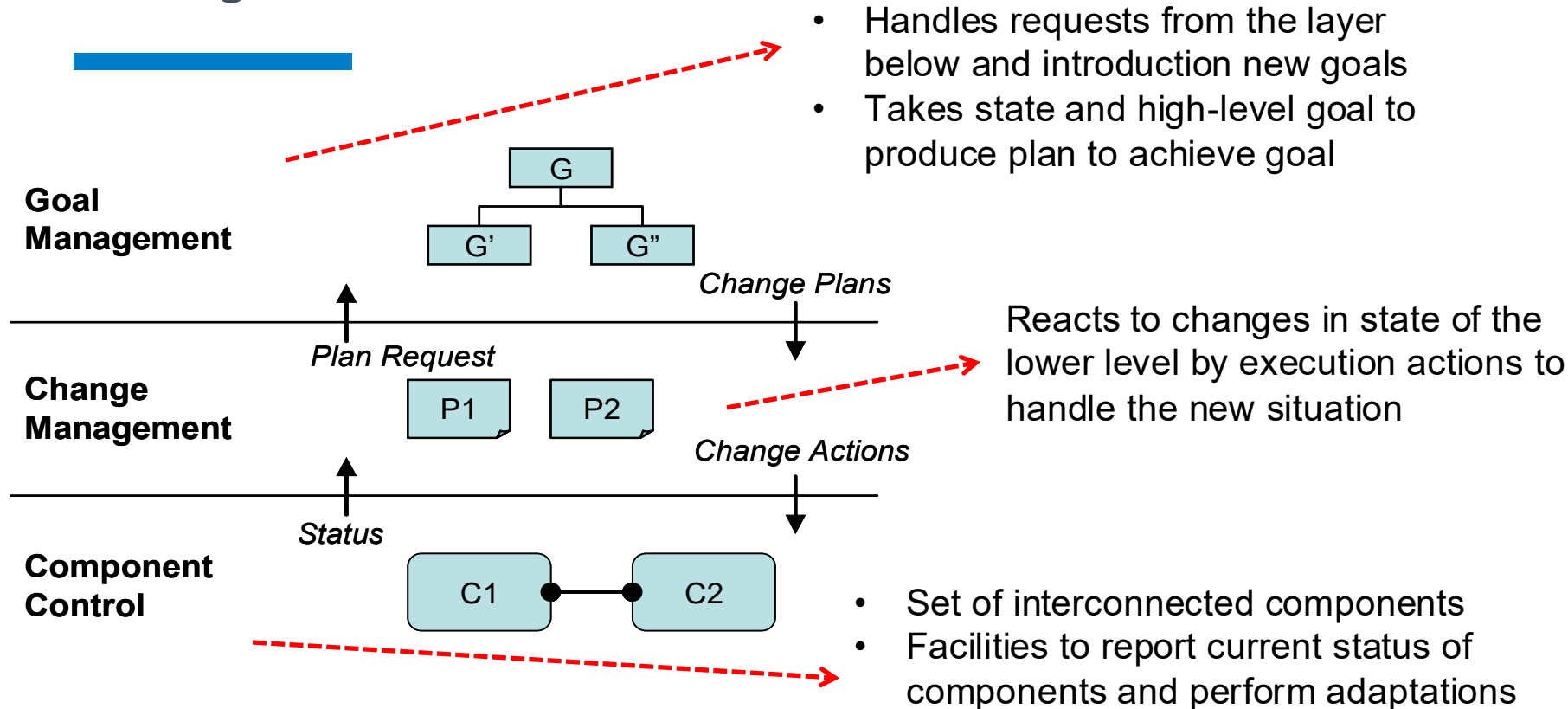
J. Kramer and J. Magee, Self-adaptation: an architectural challenge, Future of Software Engineering, 2007

E. Gat, Three-layer Architectures, Artificial Intelligence and Mobile Robots, MIT/AAAI Press, 1997

3-Layer model



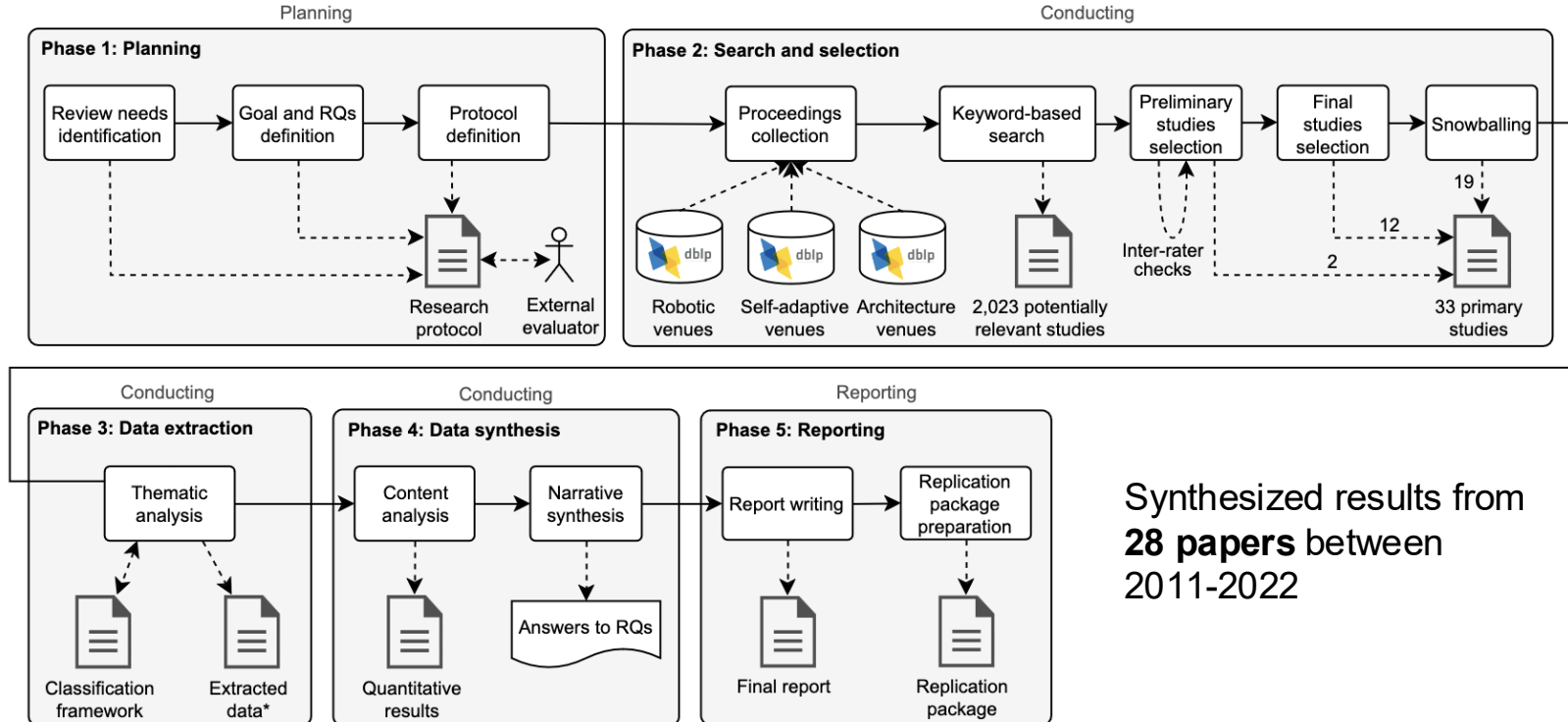
3-Layer model



Mapping existing approaches

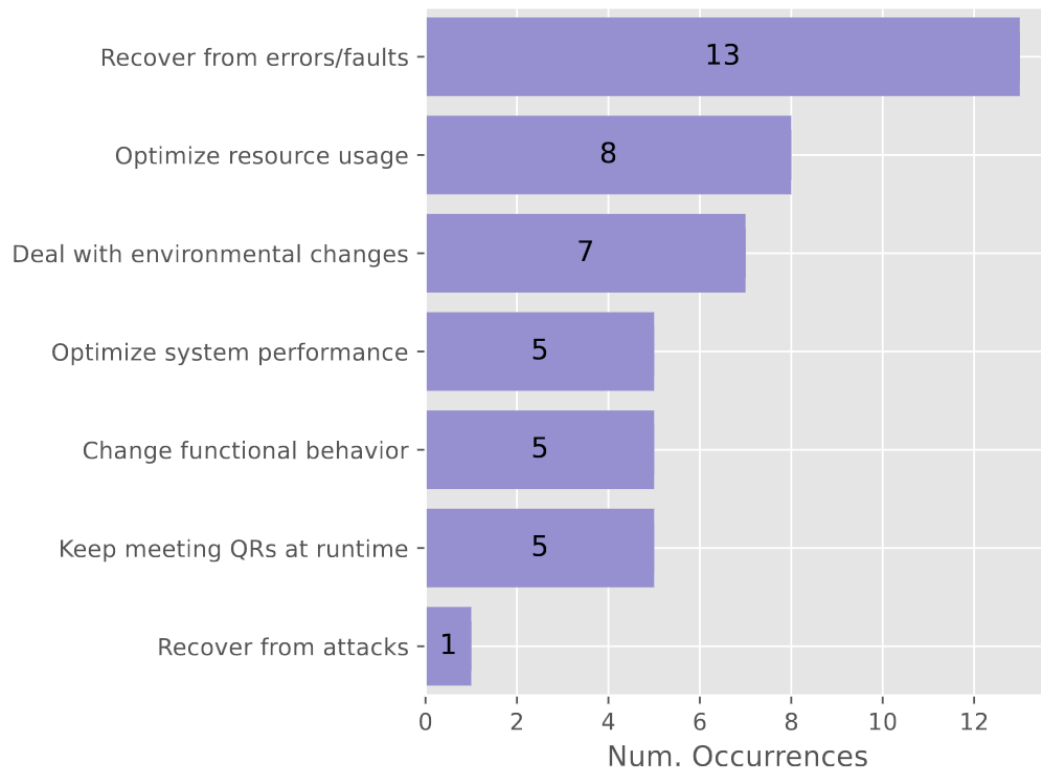
- ▣ RQ1 – What are the key characteristics of approaches for architecture-based self-adaptation in robotics software?
- ▣ RQ2 – What are the evaluation strategies of approaches for architecture-based self-adaptation in robotics software?

Study Design

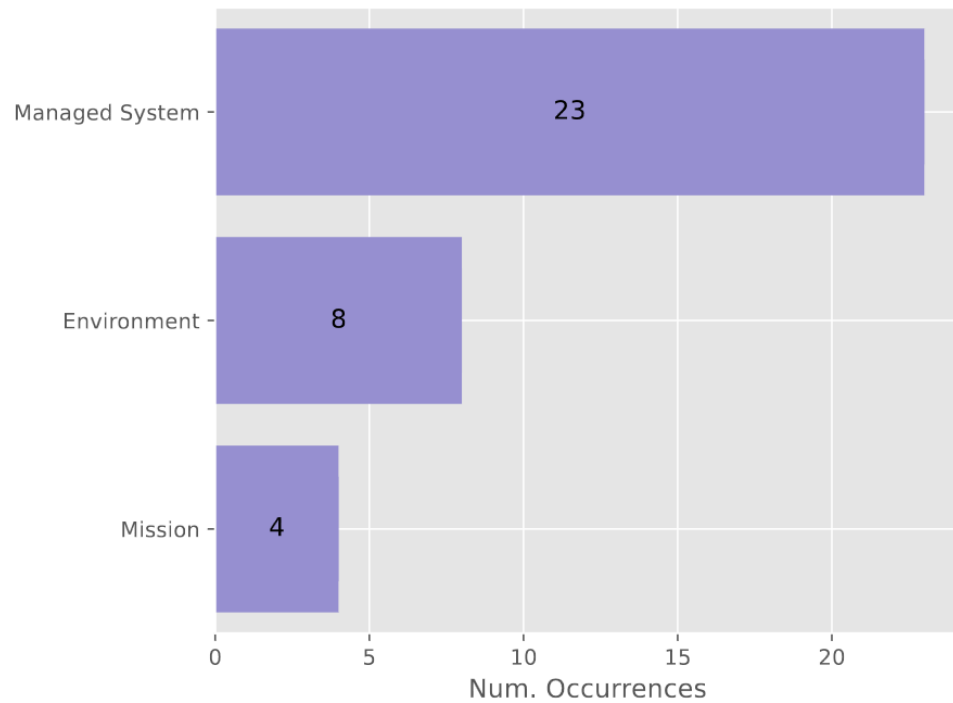


Synthesized results from
28 papers between
2011-2022

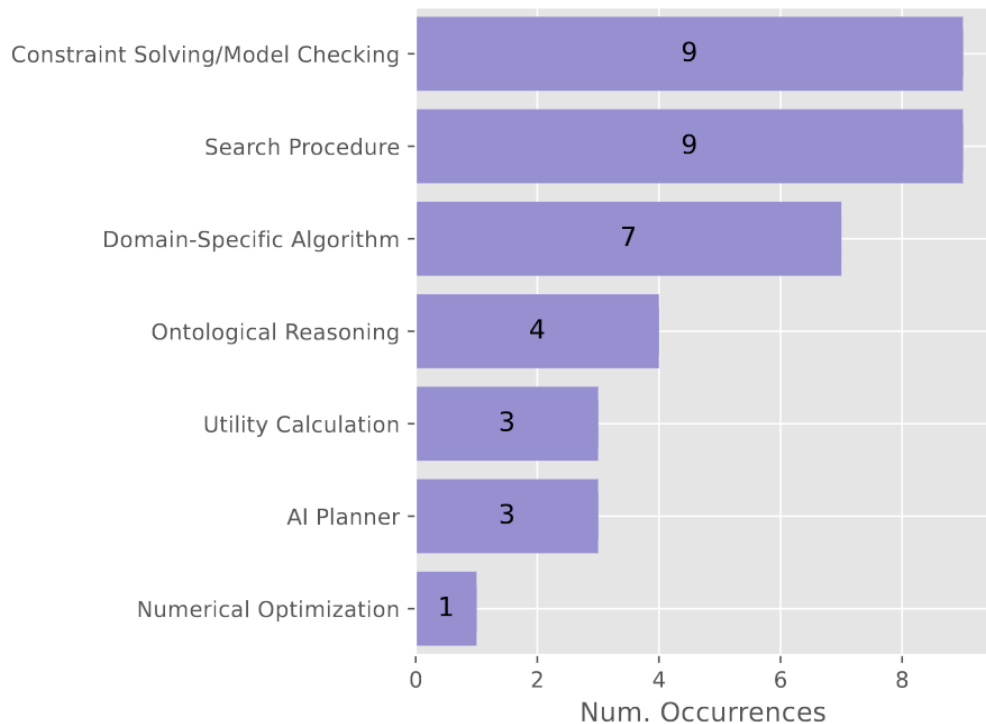
Adaptation Goals



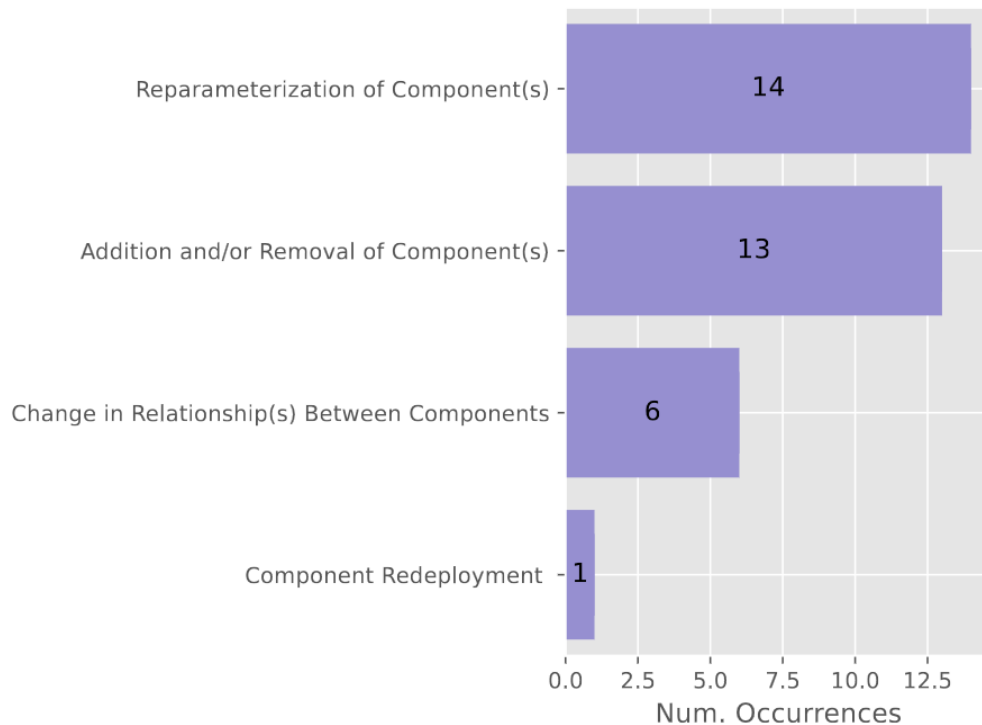
What is being sensed



Mechanisms for decision making

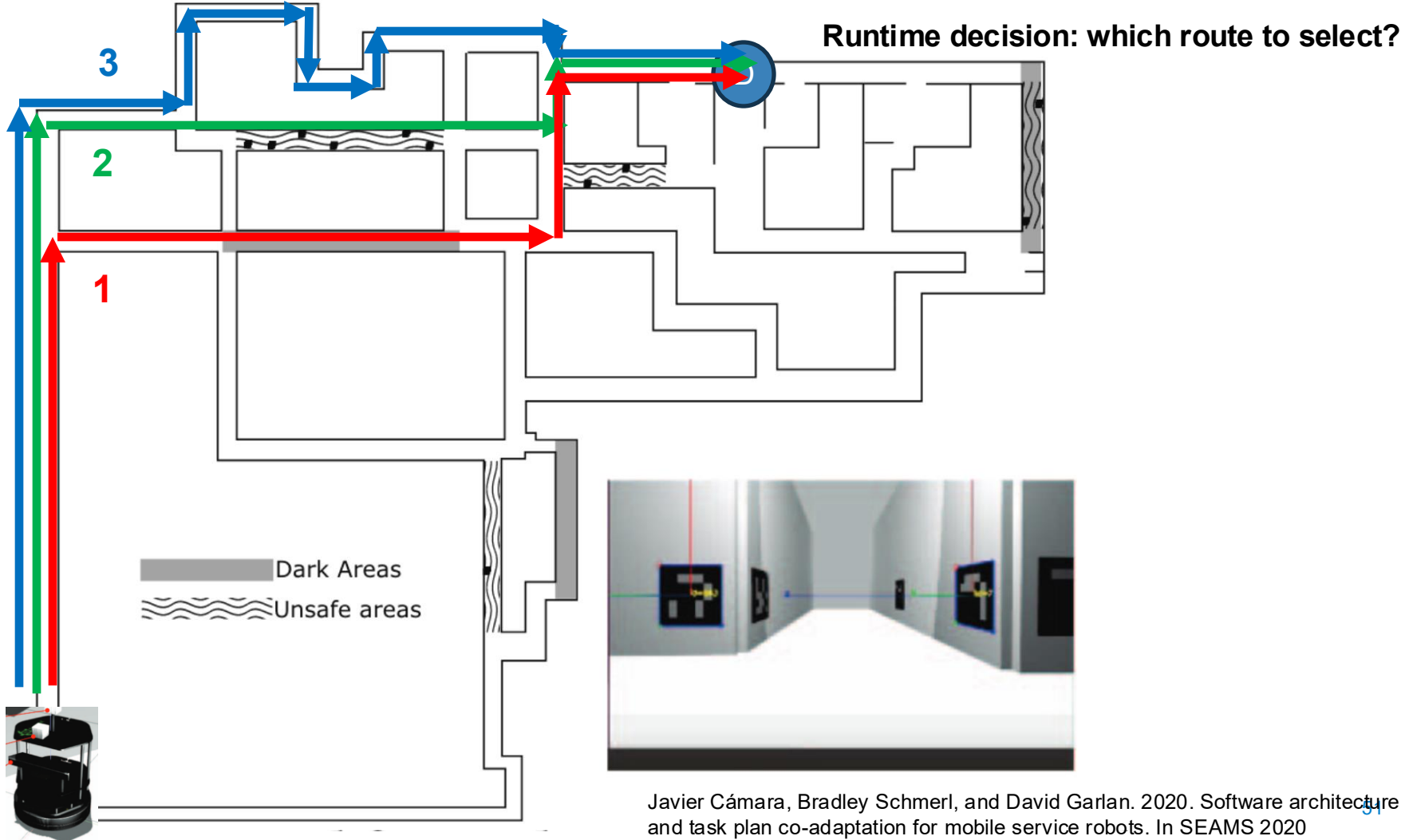


What is being changed



An approach for architecture-based self-adaption

Task and architecture **co-adaptation**



Javier Cámara, Bradley Schmerl, and David Garlan. 2020. Software architecture and task plan co-adaptation for mobile service robots. In SEAMS 2020

Runtime decision: which components to use?

Provides light in dark corridors
Terrible efficiency
Only helps with cameras

Provides 2D image of behind
Excellent efficiency
Not good in the dark
OK obstacle detection

Provides 2D planar depth field
Reasonable efficiency
Not good at obstacle detection

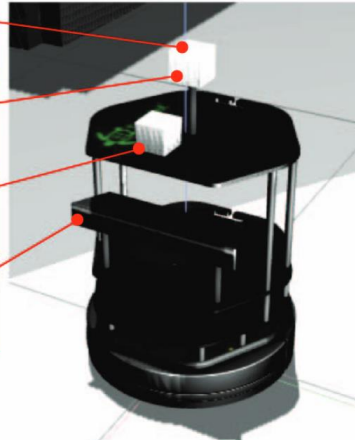
Provides 3D depth field/2D image
Excellent efficiency
Needs transform component to
convert depth image to lidar info

Headlamp: useful
in dark corridors

Back camera: images
behind the robot

Planar Lidar: depth
scans in a plane

Kinect Sensor: depth
and camera images



Category	Name	Energy cost	Accuracy	Requires
Sensing	lidar	Medium	Bad	-
	kinect	Excellent	Good	laserscanNodelet
	camera	Excellent	Medium	markerRecognizer headlamp (when dark)
Localization	amcl	Excellent	Excellent	-
	mrpt	Medium	Good	-
	aruco	Bad	Good	-
Auxiliary	laserscanNodelet	N/A	N/A	-
	markerRecognizer	N/A	N/A	-
	headlamp	Really bad	N/A	-

Concerns to consider

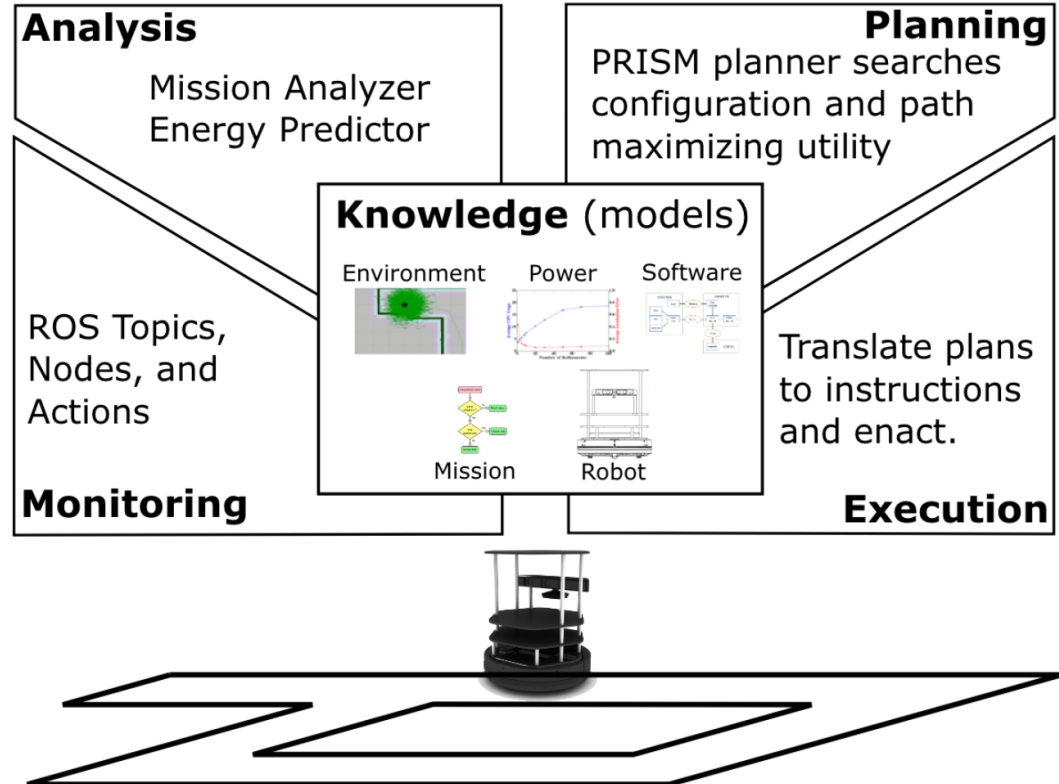
- ▣ **Timeliness** – get to the destination as fast as possible
- ▣ **Safety** – avoid obstacles
- ▣ **Energy efficiency** – minimize used energy

Approach: Model everything!

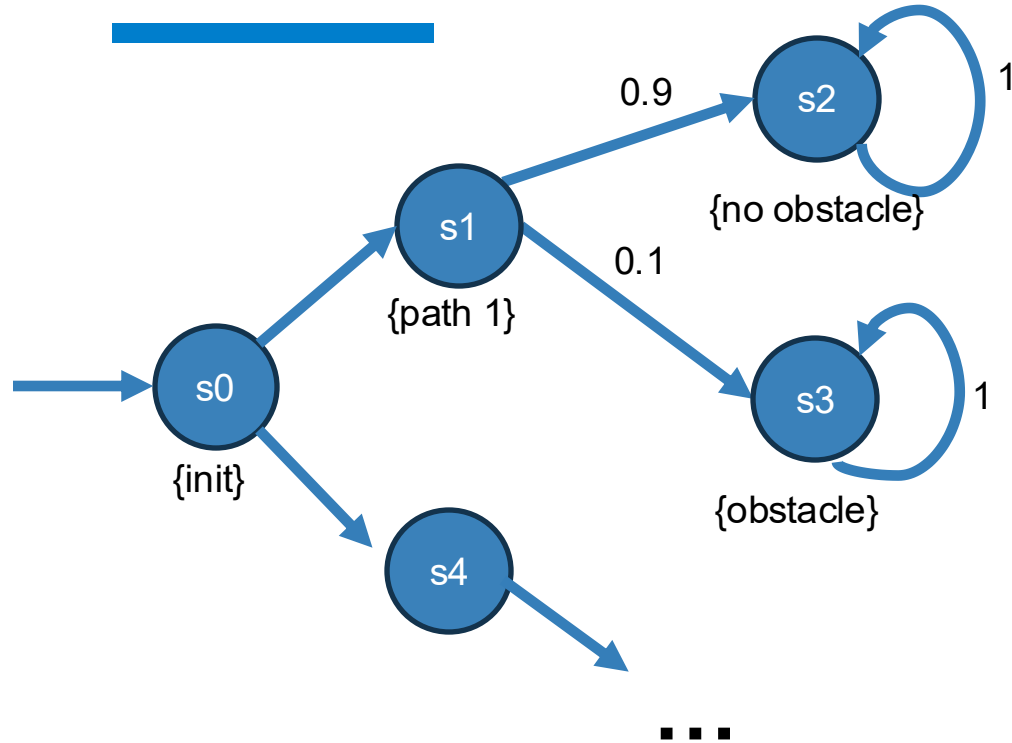
- ▣ The architecture of the robot
 - ROS2 graph and ROS2 arch. style modeled in Alloy
- ▣ The behavior of the robot
 - PRISM model (next slides)
- ▣ The resource usage
 - Energy consumption when executing a task with a certain configuration
- ▣ The map of the environment
 - Specific to the navigation task

Approach: Model everything!

- The architecture
 - ROS2 graph
- The behavior of t
 - PRISM model (ne
- The resource use
 - Energy consump
 - executing a task'
 - configuration
- The map of the e



Prism model example



module M1

$x : [0..2]$ init 0;

$\square x=0 \rightarrow (x'=1);$

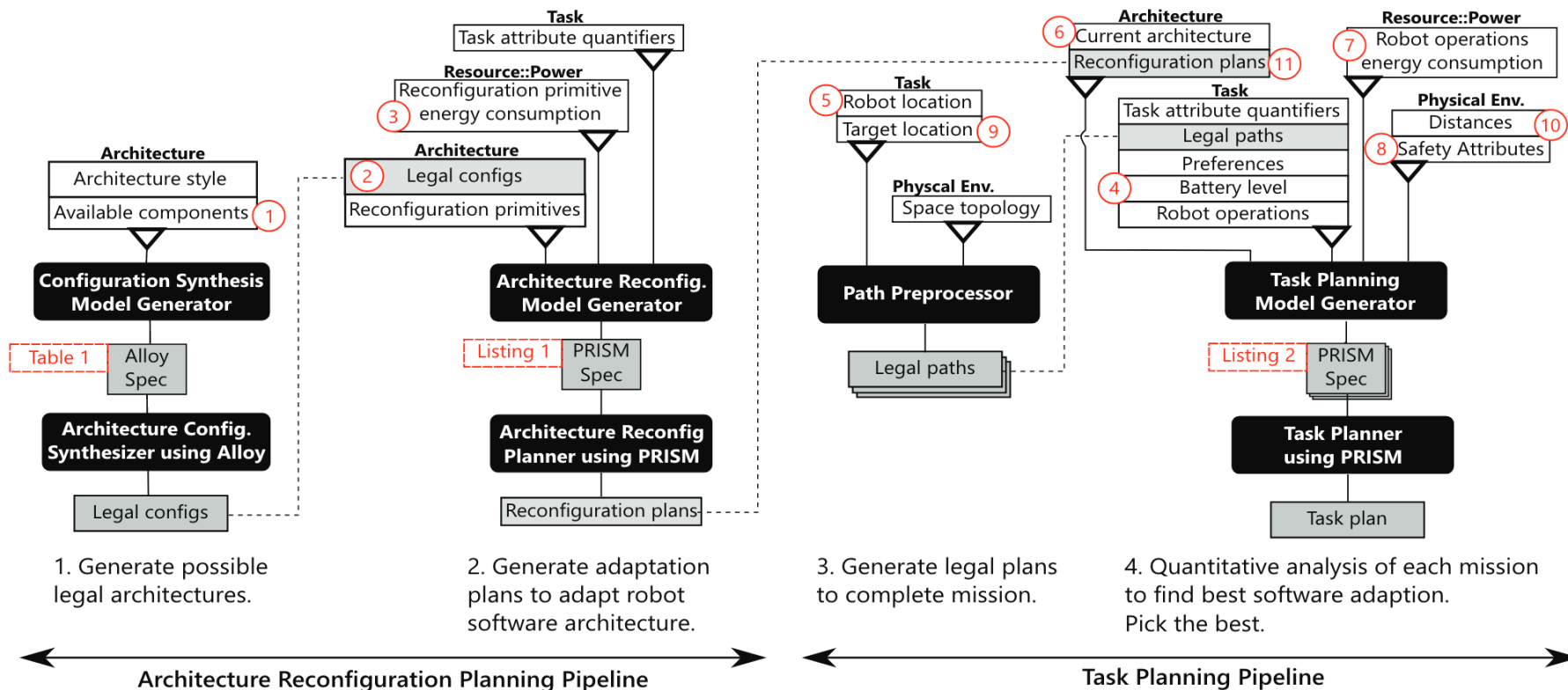
$\square x=1 \rightarrow 0.9:(x'=2) + 0.1:(x'=3);$

$\square x=2 \rightarrow (x'=2);$

$\square x=3 \rightarrow (x'=2);$

$\square x=0 \rightarrow (x'=4);$

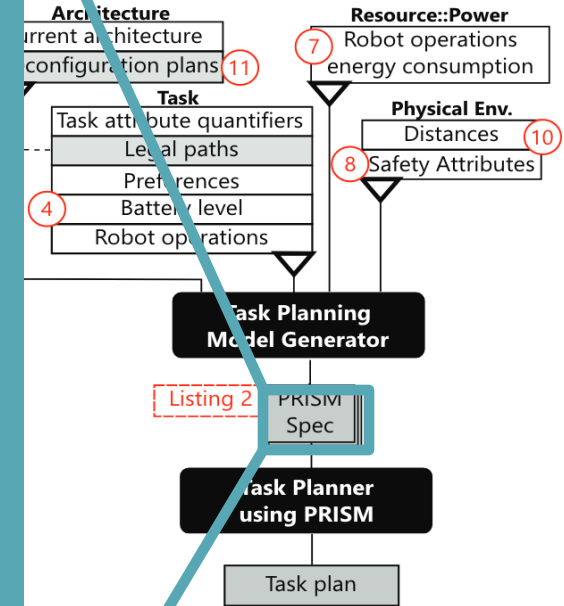
endmodule



```

1  module robot
2  b:[0..MAX_BATTERY] init INITIAL_BATTERY;④ // Task view
3  l:[0..MAX_LOCATIONS] init INITIAL_LOCATION;⑤ // Task view
4  c:[1..M] init init INITIAL_CONFIGURATION;⑥ // Architecture view
5  rd: bool init false; collided: bool init false;
6  ... // One command per legal target configuration
7  [t_set_conf_M] (c!=conf_M) & (b>MIN_BATTERY+deplete_battery_reconfM⑪
   ) & (!rd) & (!stop) -> (c'=conf_M) & (rd'=true) & (b'=b-deplete_battery_reconfM);
8  ... // One command per combination of legal config/arc among adjacent map locations
9  [lx_to_ly] (l=lx) & (!stop) & (c=conf_M) -> p_col_conf_M_lx_to_ly⑧
   : (l'=ly) & (b'=b_upd_lx_ly⑦) & (collided'=true) + 1-(p_col_conf_M_lx_to_ly):
   (l'=ly) & (b'=b_upd_lx_ly) & (collided'=false);
10 endmodule
11 formula b_upd_lx_ly= c=conf_1? max(0,b-e_lx_ly_conf_1) : ... (c=conf_M?
   max(0,b-e_lx_ly_conf_M) : 0);⑦ // One per arc between adjacent map locations
12 ...
13 const INITIAL_LOCATION;
14 const TARGET_LOCATION;⑨ // Task view
15 formula goal = l=TARGET_LOCATION;
16 formula stop = goal | b<MIN_BATTERY;
17 rewards "time"
18 [lx_to_ly] true :c=conf_1? t_lx_ly_conf_1⑩ : ... c=conf_M? t_lx_ly_conf_M :
   MAX_BATTERY; // One per arc between adjacent map locations;
19 ...
20 [t_set_conf_1] true :c=conf_2? t_set_conf_2_conf_1⑪
   : ... c=conf_M? t_set_conf_M_conf_1 : 0; ... // One per legal target configuration
21 endrewards
22 rewards "energy"
23 stop : b;
24 endrewards

```



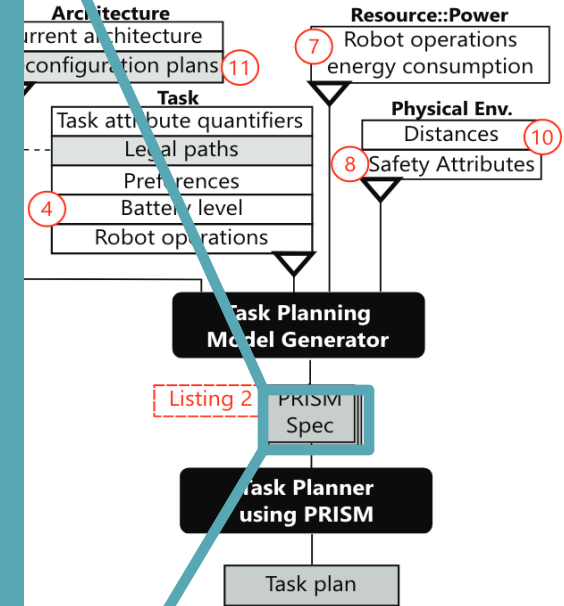
4. Quantitative analysis of each mission to find best software adaption.
Pick the best.

Task Planning Pipeline

```

1  module robot
2  b:[0..MAX_BATTERY] init INITIAL_BATTERY;④ // Task view
3  l:[0..MAX_LOCATIONS] init INITIAL_LOCATION;⑤ // Task view
4  c:[1..M] init init INITIAL_CONFIGURATION;⑥ // Architecture view
5  rd: bool init false; collided: bool init false;
6  ... // One command per legal target configuration
7  [t_set_conf_M] (c!=conf_M) & (b>MIN_BATTERY+deplete_battery_reconfM⑪
8  ) & (!rd) & (!stop) -> (c'=conf_M) & (rd'=true) & (b'=b-deplete_battery_reconfM);
9  ... // One command per combination of legal config/arc among adjacent map locations
10 [lx_to_ly] (l=lx) & (!stop) & (c=conf_M) -> p_col_conf_M_lx_to_ly⑧
11 : (l'=ly) & (b'=b_upd_lx_ly⑦) & (collided'=true) + 1-(p_col_conf_M_lx_to_ly);
12 : (l'=ly) & (b'=b_upd_lx_ly) & (collided'=false);
13 endmodule
14 formula b_upd_lx_ly= c=conf_1? max(0,b-e_lx_ly_conf_1) : ... (c=conf_M?
15 max(0,b-e_lx_ly_conf_M) : 0);⑦ // One per arc between adjacent map locations
16 ...
17 const INITIAL_LOCATION;
18 const TARGET_LOCATION;⑨ // Task view
19 formula goal = l=TARGET_LOCATION;
20 formula stop = goal | b<MIN_BATTERY;
21 rewards "time"
22 [lx_to_ly] true :c=conf_1? t_lx_ly_conf_1⑩ : ... c=conf_M? t_lx_ly_conf_M :
23 MAX_BATTERY; // One per arc between adjacent map locations;
24 ...
25 [t_set_conf_1] true :c=conf_2? t_set_conf_2_conf_1⑪
26 : ... c=conf_M? t_set_conf_M_conf_1 : 0; ... // One per legal target configuration
27 endrewards
28 rewards "energy"
29 stop : b;
30 endrewards

```



4. Quantitative analysis of each mission to find best software adaption.
Pick the best.

Task Planning Pipeline

Conclusions

Key Takeaways

- ▣ Self-adaptation can be a powerful technique for inducing robustness
- ▣ Can also be used for keeping requirements met at runtime despite uncertainty
- ▣ Co-adaptation of architecture and task can (quickly) become a complex problem
- ▣ We need (more/better) methods to handle uncertainties at runtime

References
