Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

SDP- Presentation

Adaptive Deployment of Safety Monitors for Autonomous Systems

July 13, 2020

Supervisor:
Prof. Dr. Nico Hochgeschwender
Team members:
Aswinkumar Vijayananth
Jithin Sasikumar
Manoj Kolpe Lingappa

Contents

What is the problem?

- Autonomous robots are expected to carry out challenging tasks in a trustworthy manner.
- To perform a simple task wide variety of software components are integrated with the robot such as
  - Planning
  - Perception
  - Safety critical software like monitoring, diagnosis, and fault detection and isolation
- Autonomy in robot is achieved not only by independent planning and execution of tasks but also by adaptively deploying various SW components.
- Autonomous re-deployment of safety monitoring strategies is vital.

Why is it important

- Deployment of software is an ongoing activity in robotics.
- Varying requirement by functional and safety critical modules can be full filled by redeployment.
- Hazard in robot can be eliminated by proper deployment of safety critical software components.
- Redeployment of software helps to effectively handle changing tasks, platform and environment of a robot while ensuring safety.

System Architecture

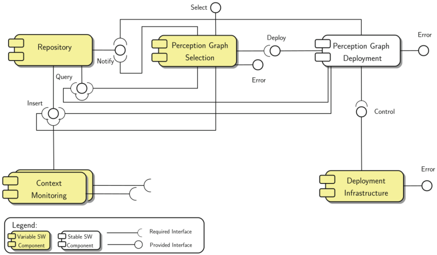

Figure 1: System diagram (Source: Hochgeschwender, Nico. “Adaptive Deployment of Safety Monitors for Autonomous Systems.” International Conference on Computer Safety, Reliability, and Security, 2019, pp. 346–3570)

- The idea of adaptive deployment is realized as a system with following components.
- **Context monitor** - responsible for observing the robot’s environment

System Architecture

- **Repository** - the central system component responsible for storing:
  - *Data generated during run-time :* current context, deployment status, dynamic platform properties like memory availability
  - *Data added during design-time :* deployment constraints, set of existing platforms
- **Safety monitors** - constantly check whether the task designated to the robot is executed in a safe manner. *(slip detectors in case of object-picking robot)*

Context	**Slip detector**
stationary-robot and gripper closed	force
stationary-robot and gripper open	tactile
moving-robot	force + tactile

Platform requirements

The list of requirement types considered during deployment planning

Requirement type	Realization
Quantity	Minimum number of tactile sensors to select a platform
Capacity	Minimum memory availability required for a deployment
Minimum	Minimum system latency
Maximum	Maximum system latency
Attribute	Presence of force sensor in a platform
Selection	Suitable force sensor type

Deployment Planning as a CSP

Variable	**Domain**
force-platform,tactile-platform, fused-platform	Set of all available platforms
force-sensor-presence	True, False
tactile-sensor-count	$Z_{\geq 0}$
memory-availability	$Z_{\geq 0}$
minimum-latency,maximum-latency	$Z_{\geq 0}$
force-sensor-type	Set of all force sensor types

The requirement Minimum number of tactile sensors to select a platform is expressed as:

"tactile-sensor-count(platform) >= min-tactile-sensor-count"

MiniZinc

- MiniZinc - a free and an open source constraint modeling language used to model constraint satisfaction and optimization problems in a higher level.

```
%Platform properties
dd =  [| 1,   1,   50, curr_memory_availability[1], 1, 10,   % PF1
       | 2,   0,  100, curr_memory_availability[2], 0, 100,  % PF2
       | 3,   0,   10, curr_memory_availability[3], 0, 1000, % PF3
       | 4,   1,    0, curr_memory_availability[4], 2, 10,   % PF4
       | 5,   1,    0, curr_memory_availability[5], 3, 10,|];% PF5
```

```
%Technical constraints

constraint force_platform[force_sensor] == force_sensor_presence;       %R1
constraint tactile_platform[tactile_sensor] >= min_tactile_sensor_count;%R2
constraint fused_platform[memory_availability]>=min_memory_fused;       %R3
constraint fused_platform[latency] <= max_latency;                      %R4
constraint fused_platform[latency] >= min_latency;                      %R5
constraint force_platform[force_sensor_type] == req_force_plfm_type;    %R6

%Platforms assigned to every slip detector need to be unique
constraint alldifferent([force_platform[name],tactile_platform[name],fused_platform[name]]);

solve satisfy;
```

Figure 2: Minizinc Code Snippet

Class Diagram

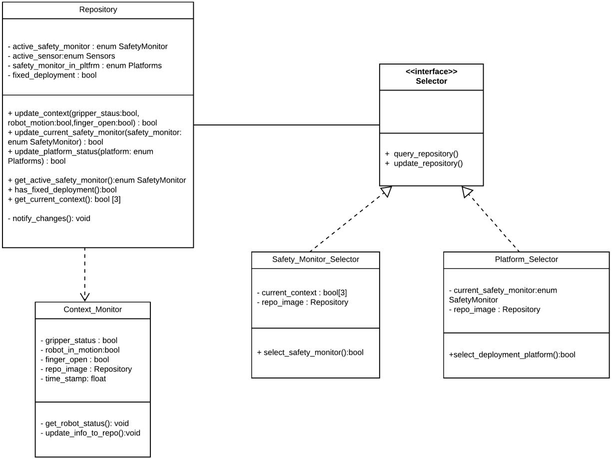

Figure 3: UML class diagram

Solving CSP using MiniZinc

- Inputs on robot's environment and memory availability of platforms at a given time step are provided explicitly.
- Mini Zinc model is invoked in the platform selector object that returns a suitable platform for deployment of the active safety monitor.
- Platform assignments are possible only if all the requirements are satisfied.
- The console outputs the selected safety monitor, platform and its properties for the current context at every time step T.

Future work

- This project only handles two of the three cases in deployment planning:
  - **Availability of exactly one matching platform**
  - **Unavailability of a suitable platform**
  - Availability of more than one matching platforms
- Selection of the most suitable platform using soft-constraints/preferences
- Use of the MiniBrass extension for including preferences in the Mini Zinc model