Mission Control

Content

Overview

Mission Control was designed to coordinate missions of heterogeneous robots in the service robot domain, while providing an architecture that should makes it easy to realize modifications in points for which the system is likely to change.

To evaluate the fitness of the architecture, we analyze a set of robot missions in the service robot domain described in the literature [1]. From the descriptions, we identified the difference and commonalities between the missions. Then we derived (i) generic mission description, i.e., a description that abstracts away the specifics of different missions, and (ii) identified points that change between different missions, these we considered point likely to change.

Starting from the analysis of a variety of missions in the service robots environment, we defined the points in the architecture that would require change either in case of integration (e.g., for communicating with a external system dispatching task requests) or in case of modification requests (e.g., the system should have a new set of priorities when allocating tasks). In the light of this analysis, we performed an evaluation of the integrability and modifiability of the architecture based on the points that are more likely to change following a questionnaire provided by Bass et. al. [2], for either quality attributes.

Mission Coordination

Each mission can be decomposed into tasks that are to be executed by specific roles within the plan. Condition in the environment can influence the mission plan. The coordinating system receives mission requests, with mission plans, involving one or more roles assigned to robots, and coordinates the executing by assigning robots to these roles. The assigned robots should be chosen between the ones available in the environment. The system should avoid failures due to low battery level by assigning only robots with sufficient batteries. The system should select only robots that have the required capabilities. If more than one robot is capable of fulfilling a role in a mission, the system should assign the one that can execute its part in less time. While evaluating the required capabilities required level of the battery, and time to execute a mission, the system may be required to consult external systems (e.g., a system that contains the map of the environment, with updated information about the availability of corridors).

Requirements

- The system should support an heterogeneous fleet, by taking into account the robot capabilities
- The system should realize coalition formation / task allocation
- The system should take into account the level of battery while allocating tasks
- The system should allocate robots according to its capabilities and resources
- The system should control the execution of local missions (within a robot, i.e.,sequencing)
- Adding new robotic behaviors (ie. skills) should be a low impact change, and
- The system should integrate robot behaviors developed by third-parties
- The system should be easy to integrate in new environments
- The system should be easy to extend for new skills
- The system should be easy to extend for new missions
- When possible, the design should limit the data exchange in the network
- When possible, the design should avoid computation on robots

Points Likely to Change

When designing the architecture, we should consider points of the system that is likely to change. We consider that the changes emerge due to changes in the environment, needed as a result of differences between missions, and across different environments. We consider that the following are points more likely to change:

- The mission plan (i.e., the sequence of tasks that each robot is required to execute. The parameters of tasks) | Between requests, between different missions

- The required set of skills required into missions | Between different missions
- The interfaces of the external systems queried during planning | Between environment, in case of change into the environment
- The available set of robots / the capabilities of each robot | Between environments, when a change occurs
- Based on the generic description we elicit the overall functional requirements of the system. Moreover, based on the likely changes, we described the architecturally-significant requirement(\ref{}) relevant for modifiability and integrability.

Main Design Decisions

- (DD.1) Hybrid between centralized and decentralized
- (DD.2) Ensembles-based Component System
- (DD.3) Task Decomposition as a Service using iHTN
- (DD.4) Simultaneous Coalition Formation / Task Allocation
- (DD.5) Coordination is extensible by descriptors
- (DD.6) Skill based implementation of robot behaviors
- (DD.7) Make the implementation independent of specific frameworks and middleware when possible = Architecture

Runtime Modules

The mission_control runtimes is divided in 6 high-level modules (i.e., Packages in Python):

- **data_model** - contains common data structures to represent missions and algorithms to operate on these data structures
- **coordination** - contains the processes for realizing mission, coordination, i.e., receive mission requests and realize coalition formation.
- **estimating** - is a submodule of coordinator, responsible for creating estimatives that will support assignment of tasks.
- **execution** - contains the processes abstractions concerned with the execution of local missions by robots.
- **deeco_integration** - creates the runtime organization using the deeco abstractions. The code in this package instantiates components from the other packages in ensemble based system abstractions, e.g., ensemble components, with knowledge base and processes, and ensemble definitions.
- **utils** - simple common functions such as logging and constants lookup.

Component Model

- **descriptors** - extend the coordination, allowing it to realize estimating for the specific tasks in the specific environment.
- **skill_implementation** - (more)

Application

An application wires all modules together (such as evaluation/experiment_gen_base/sim_exec.py), using a dependency injection framework (such as lagom).

Dependencies Between Modules

To favor the testability and modifiability of the system, we establish the following rules for what interactrions are allowed between modules:

Mission Control Runtime:

- data_model has no dependency outside core language features (e.g., basic data structures, enum, type hints, etc).
- Any module can depend on the data_model, which provides common abstractions to represent and reason about a mission.
- The coordination and execution modules do only depend on data_model and do not have dependencies between them.
- coordination module does not have static dependencies on specific skill descriptors nor environment descriptor. These are bound at initialization.
- deeco_integration depends on data_model, coordination, execution, and deeco.

Component Model

Furthermore, an application using mission_control will provide Environment Description Services, Skill Descriptors and Skill Implementations, and independent modules.

Coordination

- Environment Description Services
- Skill Descriptors

Execution

Skill Implementations

Wiring

a init script, implementing the main method, is used for instantiation of the system in a given target configuration (i.e., with a specific selection of skills and environment descriptors). These scripts are responsible for wiring together the components and it depends on every other module using the DI framework as a helper.

Evaluation

Feasibility

Main Project

hrms_mission_control_evaluation

Simulation Environment

morse_simulation

Obtained Data and Analysis

hmrs_mission_control_evaluation

Robot

robot

Review

Peer review of the architecture: Review

References

[1] M. Askarpour *et al.*, “RoboMAX: Robotic Mission Adaptation eXemplars,” May 2021.

[2] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. 2021.