23
Robot System Architectures

A robotic system is fundamentally just a collection of sensors and actuators that can interact with the environment to accomplish a set of tasks. While this definition may seem simple, the systems required to implement this definition tend to be extremely complex due to the infinite variability and uncertainty of real-world environments and the diversity among sensors and actuators. Therefore, careful and practical design of robotic systems is crucial for managing complexity, and as a byproduct enabling robust and successful robotic operations. This chapter will introduce some of the fundamental concepts, paradigms, and tools in the design of robot system architectures to enable full robot autonomy while also managing system complexity[1].

[1] D. Kortenkamp, R. Simmons, and D. Brugali. "Robotic Systems Architectures and Programming". In: *Springer Handbook of Robotics*. Springer, 2008, pp. 283–302

Robot System Architectures

The primary objective of a robotic system is to accomplish a specific set of tasks, but there are often many peripheral tasks that must also be handled to ensure the robot operates in a safe and robust way. For example a robot's goal may be to pick up objects and place them in certain locations, but in order to accomplish this task the robot should also be aware of obstacles (static or dynamic) in its environment, should be robust to sensor failures or sensor noise, and more.

Definition 23.0.1 (Robot Goal). *Complete desired tasks while monitoring and reacting to unexpected situations. Handle inputs and outputs (control/perception) from actuators and sensors in real-time*[2] *and under uncertainty.*

[2] Real-time requirements are crucial, some situations require near instantaneous reactions (e.g. less than 1 ms reaction time).

The design of the robot's system architecture is important for enabling the robot to achieve its goal without requiring extremely complex software systems for implementation. In general, the *system architecture* is defined by two major parts: the *structure* and the *style*. The structure defines the way in which the system is broken down into components, as well as how the components interact with each other[3]. Alternatively the style of the architecture refers to the computational concepts that define the implementation of the design.

[3] The structure could be represented visually as a diagram of boxes (components) that are connected by arrows (interactions).

Generally speaking there is no specific architecture that is optimal for every robotic system, but there are some paradigms that have been proven to be useful, which will be introduced in more detail in the following sections. In fact, any given system architecture may consist of multiple types of structures or styles! For a given robot, the specific choice of architecture should aim to reduce complexity[4] while not being overly restrictive and thus limit performance.

[4] For example subsystem segmentation can be useful for reusability as well as validation and unit-testing.

23.1 Architecture Structures

The architecture's *structure* defines how the system is subdivided into subsystems and how the subsystems interact. Some form of hierarchical structure is commonly used for this decomposition, which reduces complexity through abstraction (e.g. tasks at one level are of the hierarchy are composed of a group of tasks from lower-levels of the hierarchy).

23.1.1 Sense-Plan-Act Architecture

This architecture is one of the first developed, and consists of three main subsystems: sensing, planning, and execution. These components were organized in a sequential fashion, with sensor data being passed to the planner, who then passes information to the controller, who sends actuator commands. However this approach has significant drawbacks. First, the planning component was a computational bottleneck that held up the controller subsystem. Second, since the controller did not have direct access to sensor data the overall system was not very *reactive*.

23.1.2 Subsumption Architecture

An alternative to the sense-plan-act architecture that emerged not long after is the *subsumption architecture*[5]. This architecture decomposes the overall desired robot behavior into sub-behaviors in a bottom-up fashion. In this hierarchical structure the higher-level behaviors *subsume* the lower-level behaviors. In other words, the high-level behaviors can outsource smaller scale tasks to be handled by the low-level behaviors. From an implementation standpoint this architecture can be thought of as layers of finite state machines[6] that all connect sensors to actuators, and where multiple behaviors are evaluated in parallel. An arbitration mechanism is also included to choose which of the behaviors is currently activated. For example an explore behavior may sit on top of (subsume) a collision avoidance behavior, and the arbitration mechanism would decide when the exploration behavior should be overridden by the collision avoidance behavior.

[5] R. Brooks. "A robust layered control system for a mobile robot". In: *IEEE Journal on Robotics and Automation* 2.1 (1986), pp. 14–23

[6] Each finite state machine was often referred to as a *behavior*.

While this architecture is much more reactive than the sense-plan-act architecture, there are also disadvantages. The primary disadvantage of this approach is that there is no good way to do long-term planning or behavior optimization. This can make it challenging to design the system to accomplish long-term objectives.

23.1.3 Three-tiered Architecture

The *three-tiered architecture* is one of the most commonly used architectural designs. This architecture contains a planning, an executive, and a behavioral control level that are hierarchically linked.

1. *Planning*: this layer is at the highest-level, and focuses on task-planning for long-term goals.

2. *Executive*: the executive layer is the middle layer connecting the planner and the behavioral control layers. The executive specifies priorities for the behavioral layer to accomplish a specific task. While the task may come directly from the planning layer, the executive can also split higher-level tasks into sub-tasks.

3. *Behavioral control*: at the lowest-level. the behavioral control layer handles the implementation of low-level behaviors and is the interface to the robot's actuators and sensors.

The primary advantage of this architecture is that it combines benefits of the behavioral-based subsumption architecture (i.e. reactive planning) with better long-term planning capabilities (i.e. resulting from the planning level). Each of these levels will now be discussed in slightly further detail, however in practice the division among these levels is often quite blurred!

Behavioral Control Level: The components at the behavioral control level typically focus on small, localized behaviors or skills and directly interface with the robot's sensors and actuators[7]. These behaviors are typically *situated*, meaning that they only make sense with respect to a specific situation that the robot may be in. Importantly, the behavioral control components should have an awareness of the current situation (i.e. they should be able to identify if the current situation is appropriate for a specific behavior), but they are not responsible for knowing how to *change* the situation (this is left to the executive level).

[7] This layer includes algorithms from classical control theory: PID control, Kalman filtering, etc.

The tight interaction between the sensors and actuators in the behavioral control level enables a high level of reactivity in this architecture. However, high reactivity also requires that the behavioral control level not incorporate algorithms with high computational complexity. In general, the algorithms at this level should be able to operate *at least* several times per second.

Executive Level: The components of the executive level are responsible for translating high-level plans into low-level behaviors, orchestrating when low-level behaviors are executed, as well as monitoring for and handling exceptions. This component is typically implemented as a hierarchical finite state machine, but might also incorporate motion planning and decision making algorithms to break a high-level task into a sequence of smaller tasks. To orchestrate the sequence and timing for behaviors to be implemented, the executive considers temporal constraints on behaviors (e.g. whether two actions can be executed concurrently).

Planning Level: Finally, the planning level focuses on high-level decision making and planning for long-term behavior. This forward-thinking component is crucial to optimize the long-term behavior of the robot. However, the implementation of the decisions from the planner are deferred to the executive layer. In practice it might also be useful to have multiple planning levels, for example to split up mission level planning (very abstract planning) with shorter horizon planning[8].

[8] This split might be useful for computational performance reasons.

Example 23.1.1 (Office Mail Delivery Robot)**.** To further explore the components of the three-tiered robot system architecture, consider a robot whose primary task is to deliver mail within an office setting. In general, tasks that might be required of this robot include: the ability to move through hallways and rooms, avoid humans and other obstacles, open and close doors, announce a delivery, find a particular room, recharge its batteries, etc.

If a three-tiered architecture is used, the planner level would be in charge of high-level decision making tasks. For example the planner might specify the delivery order for each piece of mail to optimize the overall efficiency (i.e. by considering the relative locations of each delivery). The planner would also choose when to schedule time for recharging.

Given a task from the planner such as "Deliver package to Rm 009", the executive level would then coordinate how to accomplish the task. This might include sub-tasks such as move to the end of the hallway, open the door, enter Rm 009, announce delivery, and then wait and monitor to see if the package is retrieved. If the package is never retrieved within a specified amount of time the executive level could also choose to then carry on with the next set of tasks and send a message to the planner that the task was not completed.

Finally, the behavioral control layer would execute the tasks as specified by the executive level. This might include controlling the robot's wheels to move across the hallway, avoiding obstacles along the way. Or it could involve using a manipulator to open a door. If the current task specified by the executive was to open a door and the door was locked, the behavioral control level should eventually recognize failure and report back to the executive level.

23.2 *Architecture Styles*

In addition to choosing the robot system architecture, another very important task is to choose the architecture's *style*. An architecture's style refers to the computational structure that defines communication between components within the architecture. For example in the three-tiered architecture the style would define the method for communicating among the planning, executive, and behavioral control levels, or even between components of each individual level. The implementation of the connection style is typically referred to as *middleware*, and two of the most common architecture styles are referred to as *client-server* and *publish-subscribe*.

23.2.1 Client-Server

Middleware based on the client-server style consists of message requests from clients that the server responds to (i.e. there is a request-response message pairing). This type of connection style can also be thought of as being *on-demand* messaging. One of the disadvantages of such a messaging style is that the client typically waits for the response from the server before continuing, leading to potential deadlocks (e.g. if the server crashes).

23.2.2 Publish-Subscribe

Middleware based on the publish-subscribe style uses asynchronous message broadcasting from publishers, which can then be subscribed to by other components of the system as needed. One disadvantage of this approach is that the interfaces are less well-defined (interactions are only one-way), but the main advantage is in reliability since deadlocks cannot occur (e.g. the system is robust to missing messages or messages arriving out of order). The middleware ROS (Robot Operating System) is a very popular publish-subscribe middleware used within the robotics community today.