Subsumption architecture

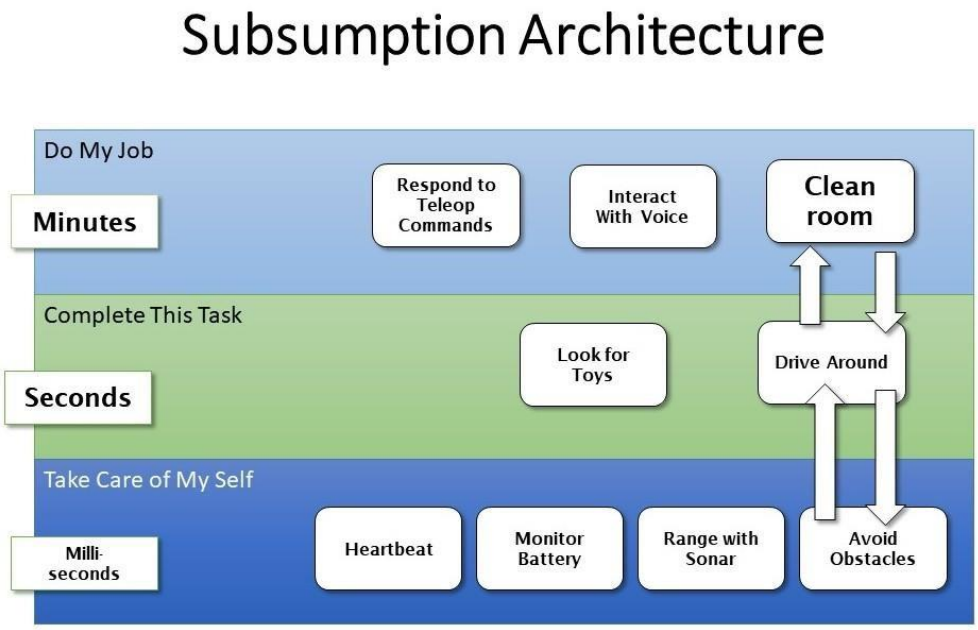
At this point, I want to spend a bit more time on the idea behind the subsumption architecture, and point out some specifics of how we will be using this concept in the design of our robot project. Many of you will be familiar with the concept from school or from study, and so you can look at my diagram and then move on. For the rest of us, let's talk a bit about this biologically inspired robot concept.

Subsumption architecture was originally described by Dr. Rodney Brooks, a professor at MIT, who would later help found iRobot Corporation and invent the Baxter Robot. Rodney was trying to develop analogues of insect brains in order to understand how to program intelligent robots. Robots before this time (1986) were very much single-threaded machines that pretty much only did one thing at a time. They read sensors, made decisions and then acted – and only had one goal at any one time. Creatures such as flies or ants have very simple brains but still manage to function in the real world. Brooks reasoned that there were several layers of closed-loop feedback processes going simultaneously.

The basic concept of subsumption has been around for some time, and it has been adapted, reused, refined, and simplified in the years since it was first introduced. What I ampresenting here is my take, or my interpretation, of how to apply the concept of subsumptionto a robot in the context of what we are trying to accomplish

The first aspect to understand is that we want our robot to act on a series of goals. The robot is not simply reacting to each stimulus in total isolation, but is rather carrying out some sort of goal-oriented behavior. The goal may be to pick up a toy, or navigate the room, avoiding obstacles. The paradigm we are creating has the user set goals for the robot and the robot determine how to carry those goals out, even if the goal is simply to move one meter forward.

The problem begins when the robot has to keep more than one goal in mind at a time. The robot is not just driving around, but driving around avoiding obstacles and looking for toys to pick up. How do we arbitrate between different goals, to determine which one has precedence? The answer is found in the following diagram:



We will divide the robot's decision-making systems into three layers. Each layer has a different level of responsibility and operates on a different time scale. At the lowest levels are what we might call the robot's autonomic nervous system – it contains the robot's internal health-keeping and monitoring functions. These processes run very fast – 20 times a second or so, and only deal with what is inside the robot. This would include reading internal sensors, checking battery levels, and reading and responding to heartbeat messages. I've labeled this level take care of myself.

The next level handles individual tasks, such as driving around, or looking for toys. These tasks are short term and deal with what the sensors can see. The time period for decisions is in the second range, so these tasks might have one or two hertz update rates, but slower than the internal checks. I call this level complete the task – you might call it drive the vehicle or operate the payload.

The final and top level is the section devoted to completing the mission, and it deals with the overall purpose of the robot. This level has the overall state machine for finding toys, picking them up, and then putting them away, which is the mission of this robot. This level also deals with interacting with humans and responding to commands. The top level works on tasks that take minutes, or even hours, to complete.

The rules of the subsumption architecture – and even where it gets its name – have to do with the priority and interaction of the processes in these layers. The rules are as follows (and these are my version):

- Each layer can only talk to the layers next to it. The top layer talks only to the middle layer, and the bottom layer also talks only to the middle layer. The middle layer can communicate with either.
- The layer with the lower level has the highest priority. The lower level has the ability to interrupt or override the commands from higher layers.

Think about this for a minute. I've given you the example of driving our robot in a room. The lowest level detects obstacles. The middle level is driving the robot in a particular direction, and the top layer is directing the mission. From the top down, the uppermost layer is commanded to clean up the room, the middle layer is commanded to drive around, and the bottom layer gets the command left motor and right motor forward 60% throttle. Now, the bottom level detects an obstacle. It interrupts the drive around function and overrides the command from the top layer to turn the robot away from the obstacle. Once the obstacle is cleared, the lowest layer returns control to the middle layer for the driving direction.

Another example could be if the lowest layer loses the heartbeat signal, which indicatesthat something has gone wrong in the software. The lowest layer causes the motors to halt, overriding any commands from the upper layers. It does not matter what they want; the robot has a fault and needs to stop. This priority inversion of the lowest layers having the highest priority is the reason we call this a subsumption architecture, since the lower layers can subsume – or take precedence over – the higher layers.

The major benefit of this type of organization is that it keeps procedures clear as to which events, faults, or commands take precedence over others, and prevents the robot from getting stuck in an indecision loop.

Each type of robot may have different numbers of layers in their architecture. You could even have a supervisory layer that controls a number of other robots and has goals for the robots as a team. The most I have had so far has been five, for one of my self-driving car projects.