```
# test connected
mcdp {  # mcdp = Monotone Co-Design Problem

    # a MCDP is defined recursively as a composition
    # of MCDPs. In this example, a "template" is a leaf
    # and indicate the interface without the implementation
    sub motor = template mcdp {
      # a motor provides a certain torque at a certain speed
      provides speed [rad/s]
      provides torque [N*m]
      # and requires $ to buy it, g to carry it,
      # and voltage, current to power it
      requires cost [$]
      requires weight [g]
      requires voltage [V]
      requires current [A]
    }

    # A "chassis" is the platform without the motors
    # and energetics
    sub chassis = template mcdp {
      # It provides a certain payload
      provides payload [g]
      # and moves it at a given linear velocity
      provides velocity [m/s]
      # It costs $ to buy
      requires cost [$]
      # We might care about the total weight (for shipping)
      requires total_weight [g]
      # It requires a motor with the given specs
      requires motor_speed [rad/s]
      requires motor_torque [N*m]
      # It also requires a controller
      # The unit "*" is a place-holder for this template
      requires control_function [any]
    }

     # the entire CDP provides the function "velocity"
     # by way of the chassis
    provides velocity using chassis

     # Constraints between motor and chassis
    torque provided by motor >= motor_torque required by chassis
    speed provided by motor >= motor_speed required by chassis

    # Motor control board in between battery and motor
      sub MCB = template mcdp {
        provides voltage [V]
        provides current [A]
         # SWAP
        requires cost [$]
        requires weight [g]
        # V, A from battery
        requires input_voltage [V]
        requires input_current [A]
      }

      # abbreviated form "<problem>.<function/resource>"
      motor.voltage <= MCB.voltage
      motor.current <= MCB.current

      # We need a battery
    sub battery = template mcdp {
        # with the given capacity
        provides capacity [J]
        # supplying a certain voltage/max current
        provides voltage [V]
        provides current [A]
        # it will cost money
        requires cost [$]
        # and need to be carried
        requires weight [g]
      }

  # "Autonomy" is a placeholder. It provides
  # a control function and requires SWAP resources.
  sub autonomy = template mcdp {
    # See paper "Resource-Aware Robotics-Application"
    # for a discussion of how to define a partial order
    # on the set of controller and computation graphs
    provides control_function [any]
    requires computation_graph [any]
    requires cost [$]
    requires weight [g]
  }
  autonomy.control_function >= chassis.control_function

  sub computer = template mcdp {
    # a computer is something that runs a program
    # defined by a computation graph
    provides computation_graph [any]
    # and needs cost+SWAPto do so
    requires voltage [V]
    requires current [A]
    requires cost [$]
    requires weight [g]
  }
  autonomy.computation_graph <= computer.computation_graph

  # Co-design constraint: we must carry everything
  chassis.payload >= (battery.weight + MCB.weight
    + autonomy.weight + computer.weight + motor.weight)

  # Co-design constraint: we must have enough energy on board
  # to last for the duration of the mission
  provides endurance [s]

  # sum current of components
  current = MCB.input_current + computer.current
  # take the maximum voltage (conservative)
  voltage = max(MCB.input_voltage, computer.voltage)

  # Watts = Amperes * Volts
  power = current * voltage
  # Joules = Watts * seconds
  energy = endurance * power

  # Requirements for the battery
  battery.capacity >= energy
  battery.current >= current
  battery.voltage >= voltage

  # We can take into account the shipping cost
  sub shipping = abstract mcdp {
    provides ships [g]
    requires postage [$]

    # the shipping rate depends on the destination
    rate_continental_US  =     0.5 $ / lbs
    rate_low_earth_orbit = 10000.0 $ / lbs
    rate = rate_continental_US

    # postage proportional to weight
    postage >= rate * ships
  }

  shipping.ships >= chassis.total_weight

  # What do we need to minimize overall?
  # 1) Minimize the cost of everything
  requires cost [$]
  # cost of building
  components_cost = (chassis.cost + motor.cost
              + battery.cost + MCB.cost
            + autonomy.cost + computer.cost )
  # plus cost of shipping
  cost >= components_cost + shipping.postage

  # 2) Also minimize the battery weight
  requires w >= battery.weight

}
```