# OO Analysis and Design Example: Microwave Oven

# Example: A Microwave Oven Simulator

Requirements Statement:

A microwave oven heats food by bombarding it with microwaves. The hungry user pushes buttons on a keypad to tell the oven how long to heat the food, and at what power level. The keypad also has a button for starting the oven. When the timer times out, it stops the oven and rings the bell. The microwave will also stop if the door opens. Assume that the microwave generator works by controlling a Klystron with a one second duty cycle.

- *Find the classes...*

# Microwave Oven - Use Cases

1) Set Time
   - User inputs a Number, then presses the Set Time Button.
   - System updates Display for time left.
   - System sets Timer.
2) Set Power Level
   - User inputs a Number, then presses the Set Power Level Button.
   - If the number is not in (1-10) then do nothing.
   - System updates Display for power level.
   - System sets microwave Generator.
3) Open Door
   - Stop Timer from counting down.
   - Disable Generator from generating microwaves.

# Microwave Oven Use Cases

4) Close Door

Enable microwave Generator.

5) Push Start button

If Door closed:

Begin counting down.

Generate microwaves, while time left > 0.

Keep Display up to date with time left.

Stop and ring Bell when time = 0.

# Microwave Oven – Classes (analysis)

- Oven
- Generator
- Timer
- Clock
- Keypad
- Start Button
- Door
- Bell
- Other Buttons:
  - Clear, [numbers] 0-9, Set Time, Set Power Level
- Klystron
- Display

# Microwave Oven: Architecture

- There are three types of events in the user interface:
  - buttons being pushed.
  - the door being opened or closed.
  - timing events from a clock chip.
- We do not wish to continuously poll the Door to see if it is open or closed, nor the Timer to see how much time has gone by; an alternative is for the Oven to be notified when the Door is opened or closed, and when the Timer ticks.
- Use an ***event-driven***, rather than a ***polling*** architecture.
- Assume that the events of interest cause methods to be invoked on your classes. Don't worry about how this works for now.

# Microwave Oven: Architecture

- We wish for all of the components of the Oven (the Timer, the Keypad, etc.) to be reusable for an entire product family of ovens.
- The Button classes will not have responsibility for system control. We will keep the Button classes simple, with a single uniform interface function: push(). To have a reusable design, we do not want the buttons to know about all the components in the system.

# Microwave Oven: Responsibilities

Oven:

- Know the state of the system.
- Delegate button commands to aggregated objects.
- Get notified when the Door gets opened or closed.
  - » Disable Generator.
- Get notified when the Timer has counted down to 0.
  - » Disable Generator.
  - » Ring Bell.
- Get notified when the Timer has ticked.
  - » Update Display.
  - » Notify Generator.

# Microwave Oven: Responsibilities

Microwave Generator:

- Know power level.
- Generate microwaves at power level, if enabled.
- Get notified of clock tick events.
  - » Activate the Klystron for a varying percentage of the microwave generation duty cycle, depending on the power level (integer between 1 and 10).

Timer:

- Know time left.
- Count down.
- Notify the Oven of the following events:
  - » The clock has ticked (every tenth of a second).
  - » The time left has reached 0.

# Microwave Oven: Responsibilities

Display:

- Show time left.
- Show power level.
- Know and show current number being input into Keypad.

Keypad:

- Be a container class for the Buttons.

Door:

- Notify the Oven of open and close events.

Bell:

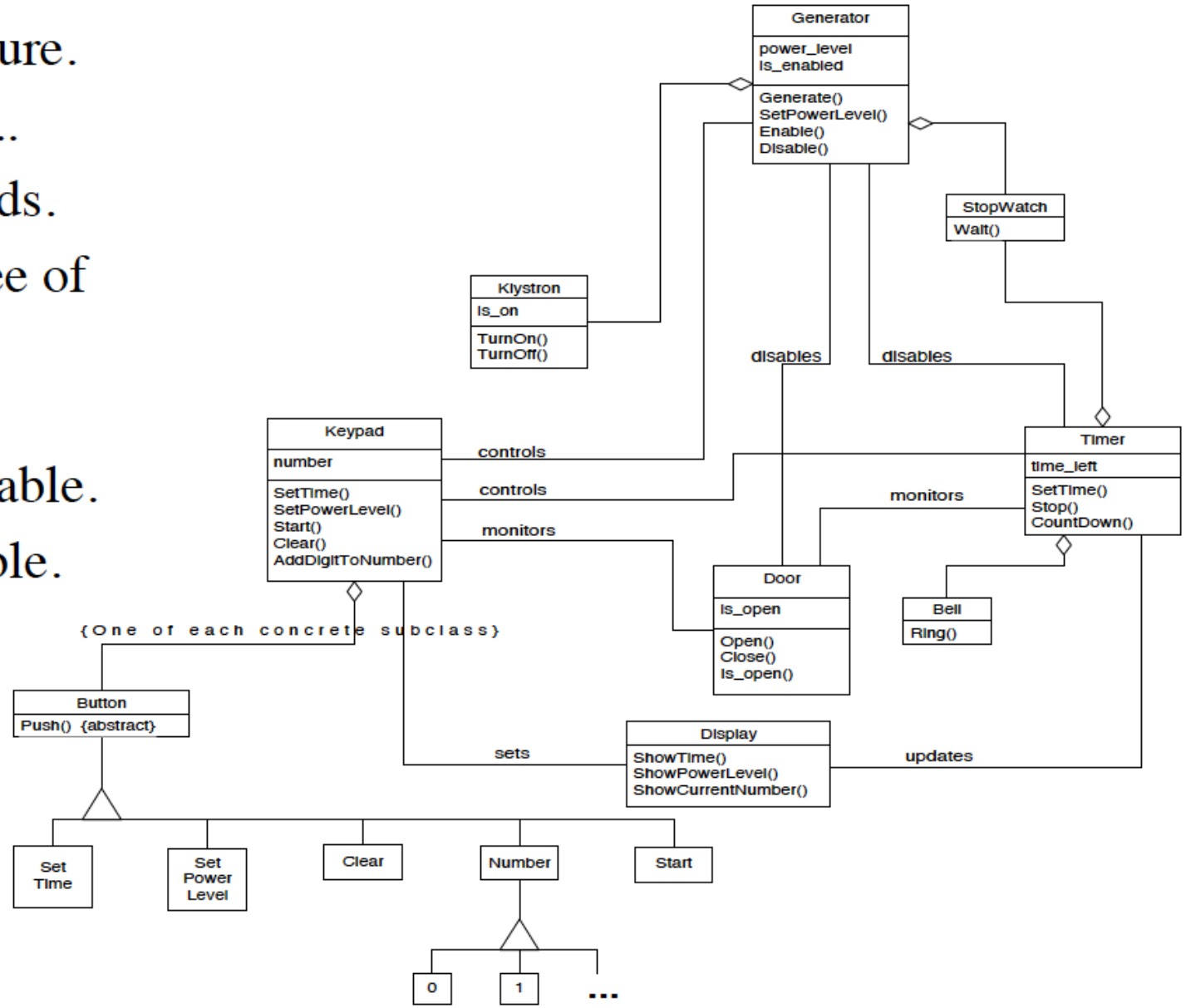
- Ring.

All Buttons:

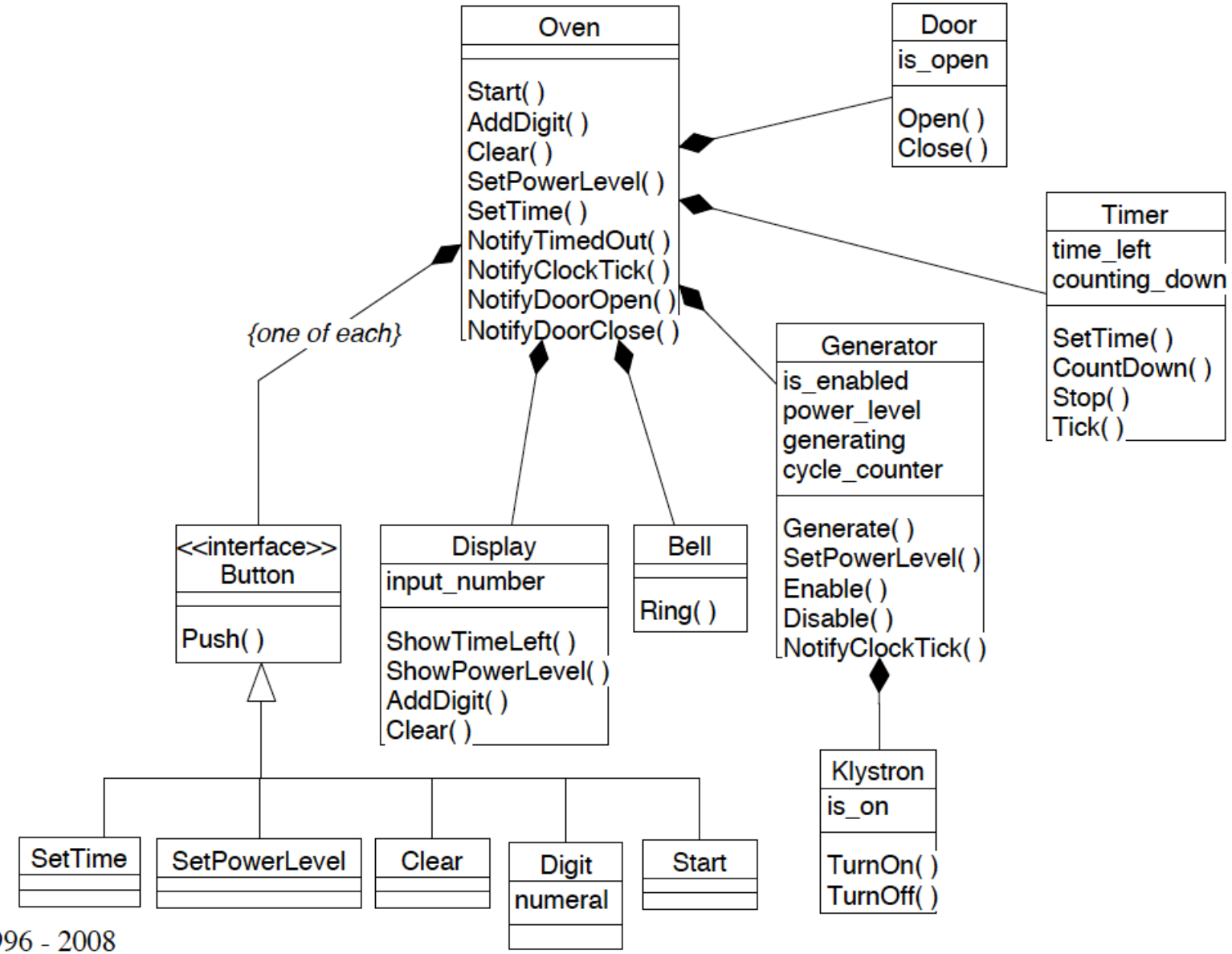
- Get pushed; send a command to the Oven.

# The Microwave Oven Class Diagram #1

- Polling architecture.
- Works fine, but...
- Requires 2 threads.
- Has a high degree of object coupling.
- Complicated.
- Keypad not reusable.
- Timer not reusable.
- ***REFACTOR*** ...



# The Microwave Oven Class Diagram #2



# Comparison of the 2 Microwave Oven models

- Model 2 has an Oven class (an example of the ***Mediator*** design pattern). This simplifies collaborations between components, which allows them to vary independently because they are now loosely coupled.
- Model 2 better resembles the ***event-driven*** problem.
- Model 2 does not have a Keypad class, as such a class would not have any responsibilities. In Model 1 however, it does have responsibilities.
- Models 1 & 2 also differ in the way they treat the 10 digits.
- Model 1 has a StopWatch class.
- Model 2 is both simpler and more flexible!

# Design Pattern: *Mediator*

***Intent*:** Encapsulate the interaction(s) between a set of classes.

***Examples*:**

- The Oven
- The Sticks Game Referee.

***Applicability*:**

- Whenever the communication between classes gets complex.
- Whenever two or more interfaces must vary independently.
- Whenever information must flow between two classes, but neither class wishes to accommodate the other.

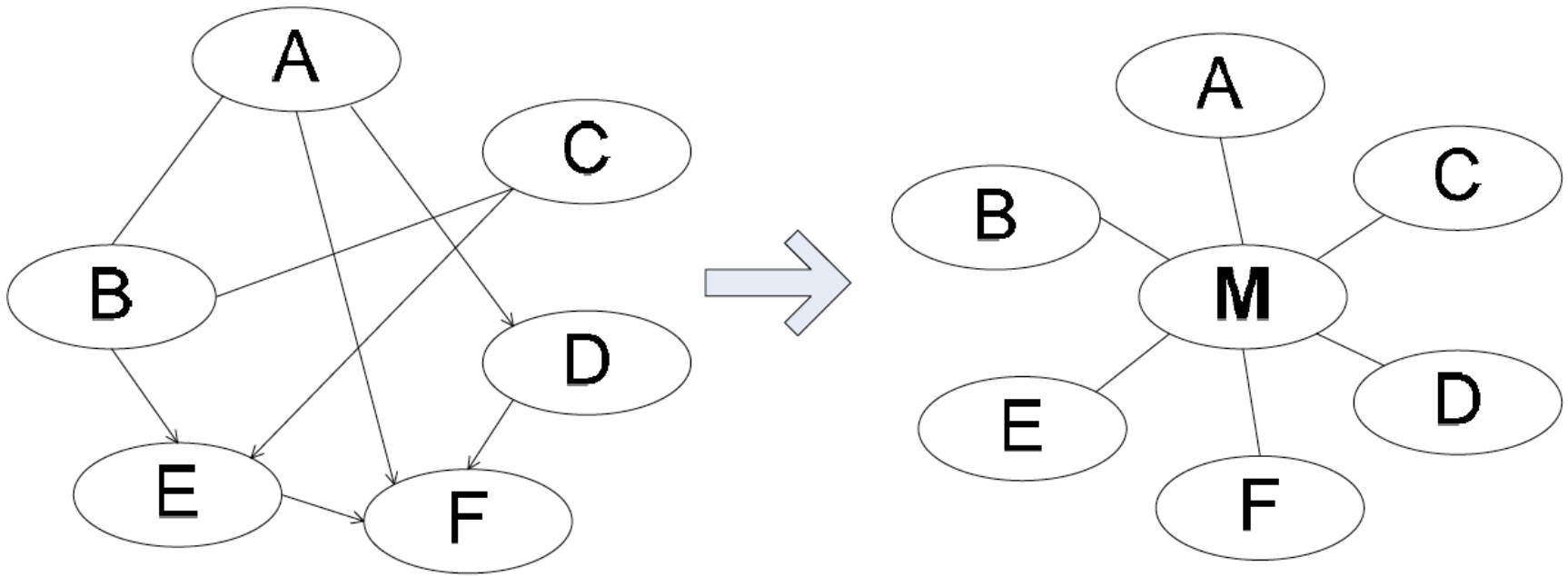
***Pros*:**

- Promotes loose coupling between objects.
- Simplifies complex collaboration diagrams.

***Cons*:**

- The Mediator itself can become quite complex.

# Mediator according to Graph Theory




N.B.: This diagram is not UML

# Microwave Oven Sequence Diagram