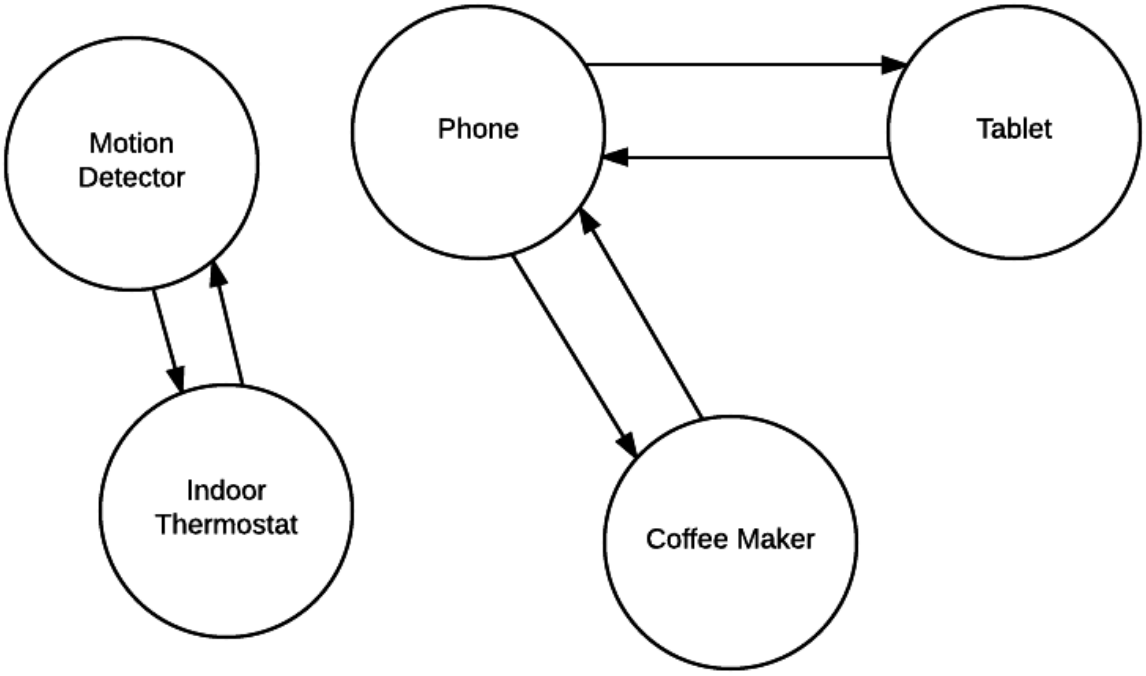
Software Design & Architecture

Mediator Pattern

Imagine that you want the house of the future. You want your house to change its own temperature once you have left, to brew your coffee when the alarm on your phone goes off, and to load the latest Globe and Mail news issue onto your tablet if you're home and it's Saturday morning.

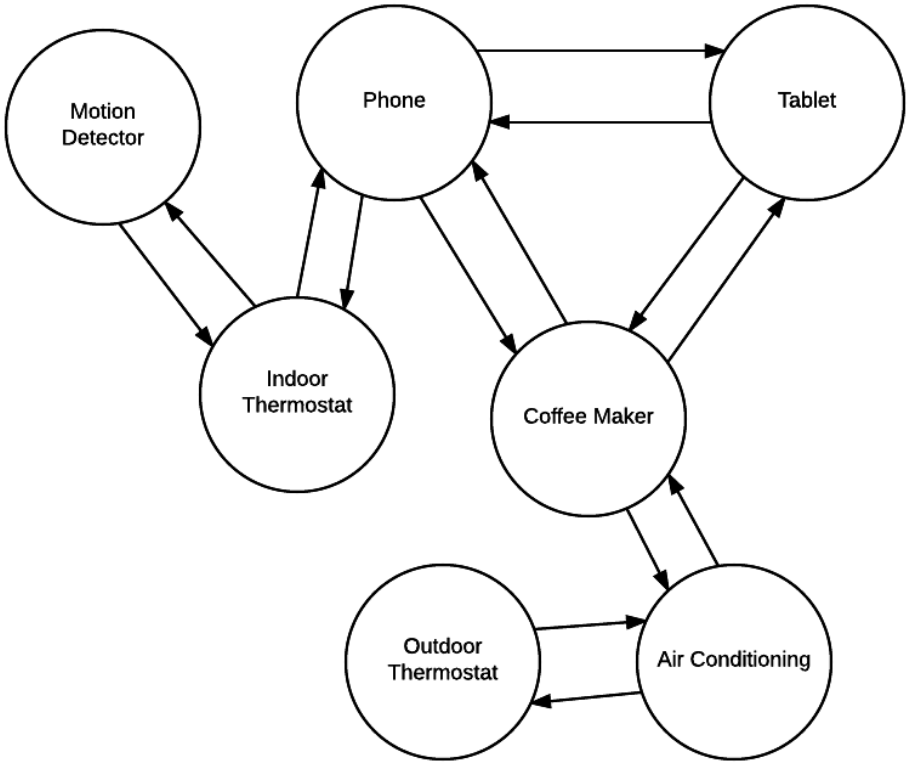


This plan starts out nicely; you are an accomplished software developer, so making these different objects talk is no big deal to you.

v.20170825

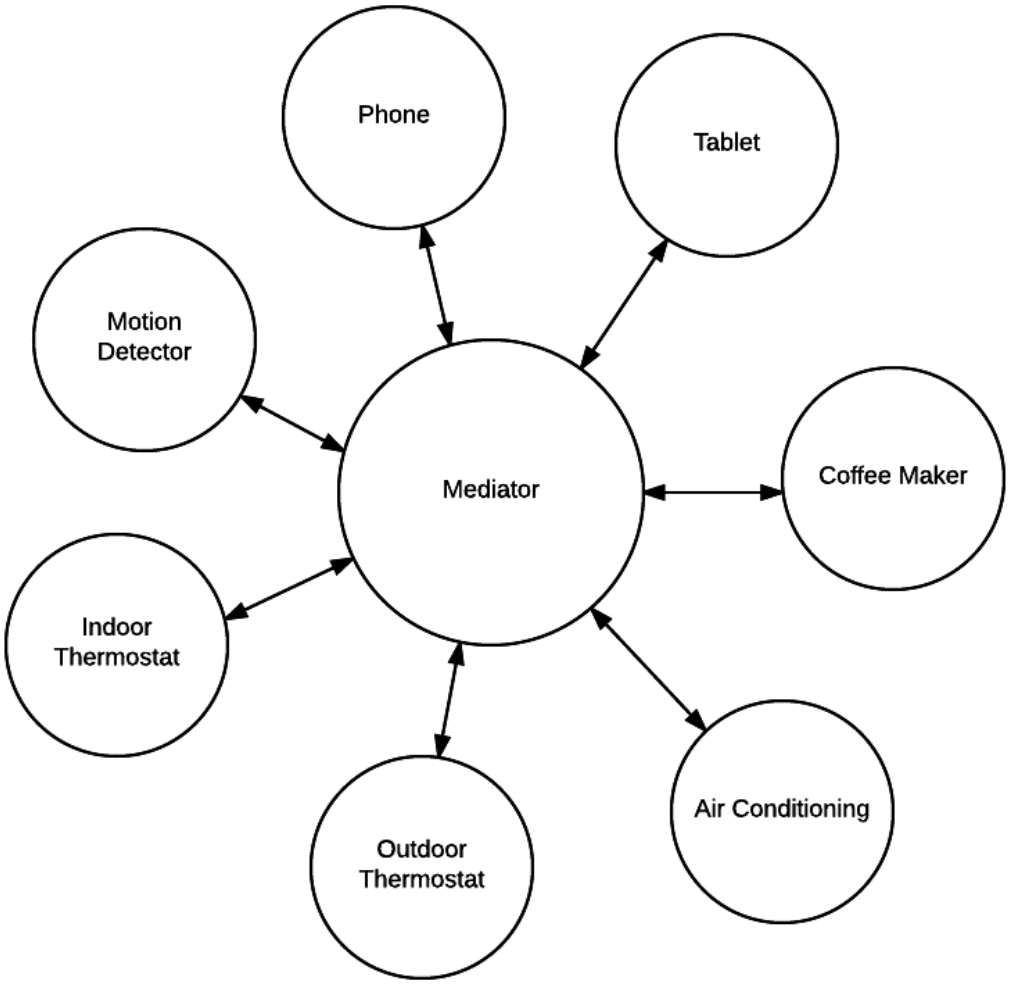
UNIVERSITY OF ALBERTA
FACULTY OF SCIENCE

You love your futuristic house, and you keep adding more rules and more devices. Eventually you realize that this tangle of pairwise interactions - interactions between two objects - is becoming complicated and difficult to maintain:



Looks like you need a **Mediator Pattern.**

In the Mediator pattern, you will add an object that will talk to all of these other objects and coordinate their activities. Now, instead of objects being engaged in various pairwise interactions, they all interact through the mediator.

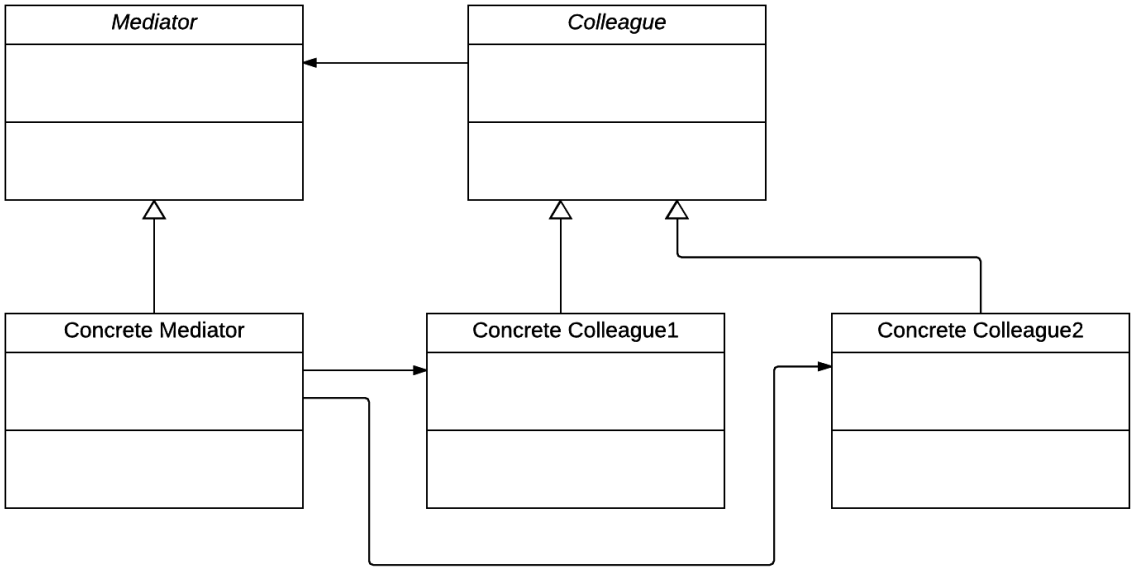


The communication between an object and the mediator is two-way: the object informs the mediator when something happens. For example, your Phone object could inform the mediator when your alarm goes off.

The mediator can perform logic on these events. For example, the mediator could track the number of times the alarm has gone off.

Finally, the mediator can request information or behaviour from an object. If the alarm has gone off three times, then the mediator could ask your door to open, which would allow your dog to go in and jump on you to help wake you up.

IMPLEMENTATION



The objects associated with your mediator are called colleagues. You define an interface for the interactions between the mediator and colleagues, then instantiate a concrete mediator and concrete colleagues as necessary.

The communication could be implemented as an Observer pattern. Each Colleague is a subclass of the Observable class, and the Mediator is an Observer to each of them. In this case, the colleague

should pass itself as a parameter to the mediator, so that the mediator knows to check that colleague instead of checking all of them. The communication could also occur through an event infrastructure.

USAGE IN SOFTWARE

A common usage for the mediator pattern, identified by the Gang of Four, is for dialog boxes with many components. As the user makes selections, such as checking a box or choosing a certain bullet out of a list, other parts of the dialog may have to be grayed out or changed in some other way. Instead of components talking to each other directly, a mediator can easily manage the different interactions like this.

CONCLUSION

The Mediator allows for loose coupling between colleagues. Since each of these colleagues only communicates with the mediator, they can be reused more easily. Centralizing the logic of interaction between all of these related objects in one central object makes for code that is easier to read, maintain, and extend.

On the other hand, the mediator can quickly become very large. Large classes are generally discouraged because they make code more difficult to debug. The benefits of centralization should be balanced against the downsides of a large, difficult-to-read mediator class.