Ingeniería de Software I

Pauta - Examen Final

Departamento de Sistemas – FIET- Universidad del Cauca

Parte A. Caso de Desarrollo 1 (Calentamiento)

Se desea implementar relojes con alarmas, a partir del reloj definido y modelado en clase. En el siguiente diagrama se muestra el diseño y se ha agregado la clase Alarma, la cual aún no se ha integrado a la solución.

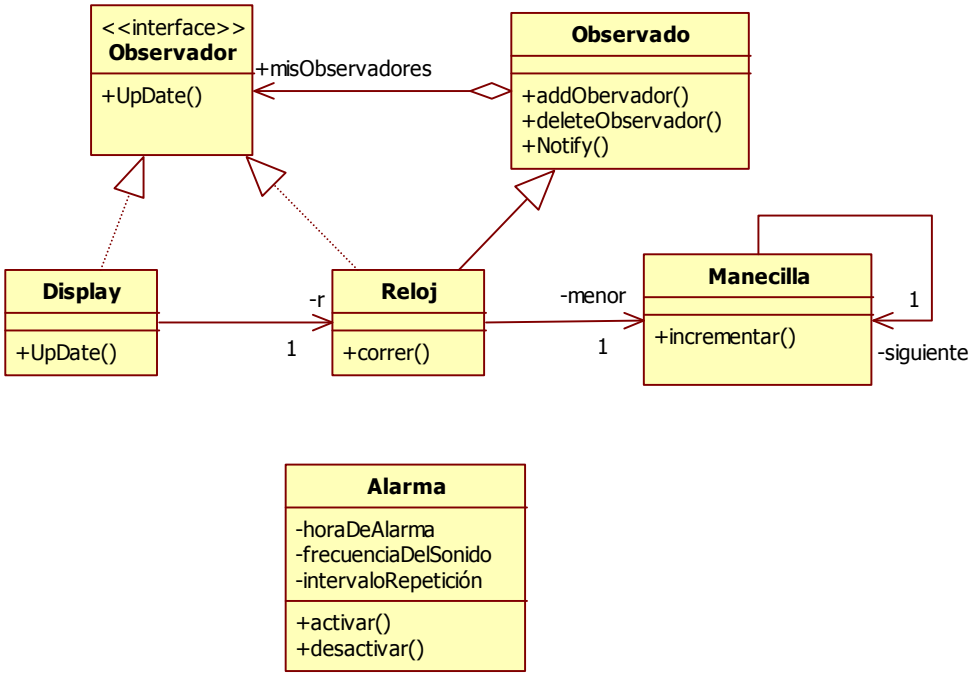
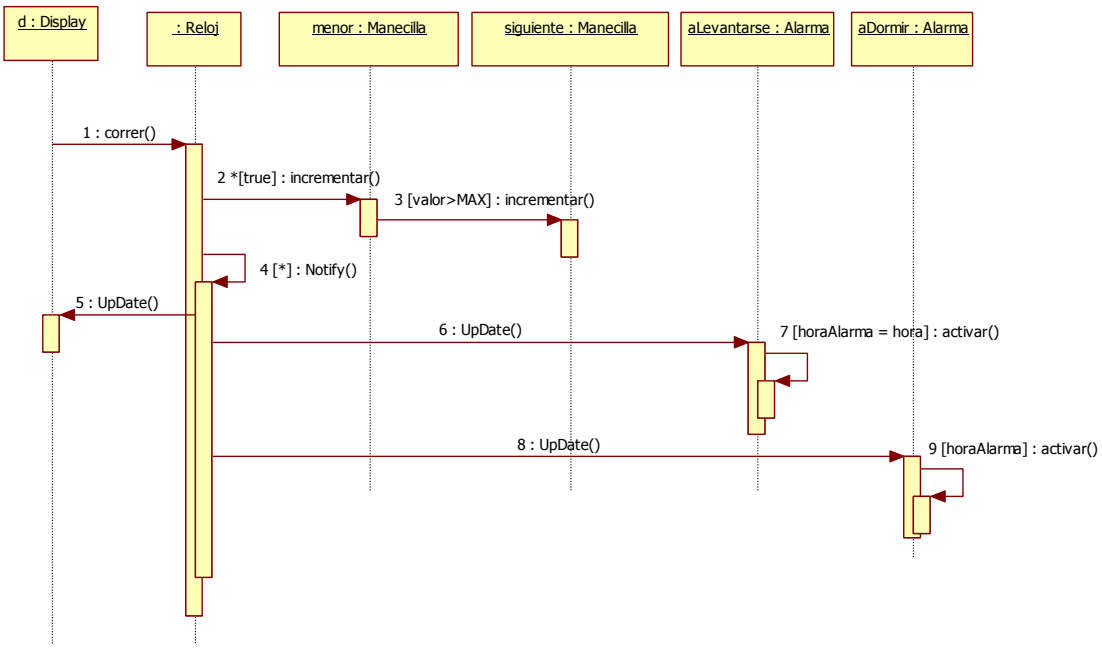
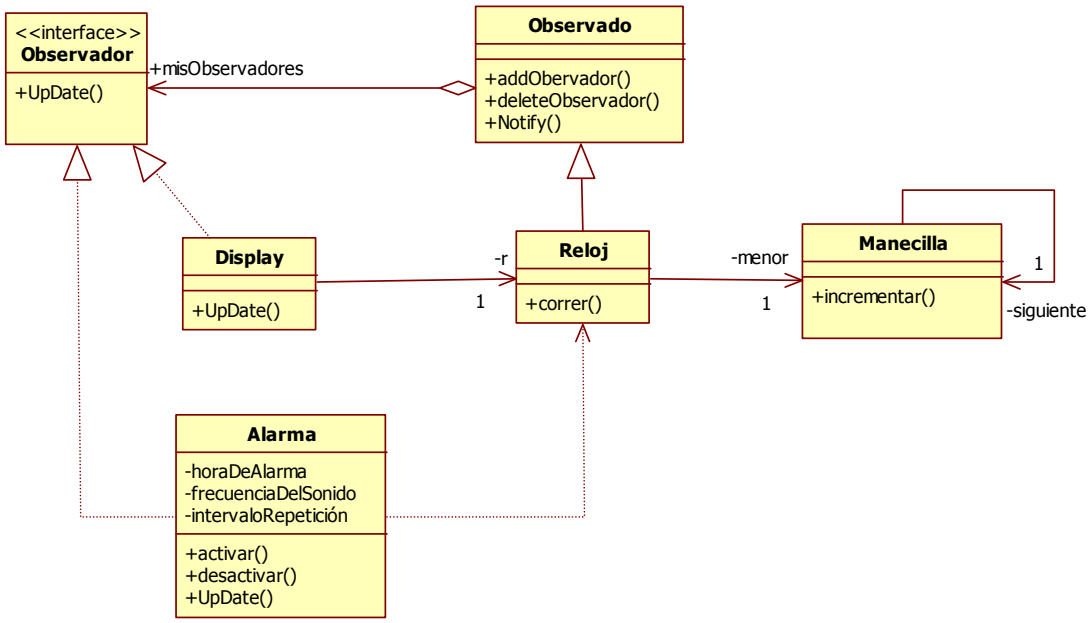


Figura 1. Diagrama de clases del sistema del Reloj y la Clase Alarma aún sin integrar

1. Con el fin de integrar la clase Alarma al resto del diseño:
    a. (0.5) Haga un diagrama de secuencia que permita comprender la forma en que dos objetos de alarma interactuarían con el resto del sistema, cuando el reloj alcanza sus horas de activación. Los objetos alarma serán notificados de la hora de la misma forma que el Display, es decir tendrán el mismo rol de observadores del observado(reloj). Cuando en la alarma se llama al método UpDate se puede evaluar si es la hora de activar la alarma y si es el caso se activa.



b. (1.0) Haga un diagrama de clases que muestre el nuevo modelo integrándole la clase Alarma. Diga en qué principio de diseño se basó y explíquelo. Gracias a que el diseño ha sido logrado partiendo del principio "programe para una interface y no para una implementación", es posible agregar la alarma como uno de los roles ya definidos (observador) y por tanto a las clases relacionadas con este comportamiento no se verán afectadas.



c. (0.5) Implemente en su lenguaje de programación favorito y por completo la clase Alarma modificada de acuerdo a su incorporación al diseño. Excluya la implementación interna de los métodos activar y desactivar.

```
public class Alarma implements Observador{
        private String horaDeAlarma;
        private int frecuenciaDeSonido;
        private int intervaloRepeticion;
public Alarma(String h, int f, int r){
        horaDeAlarma = h;
        frecuenciaDeSonido = f;
        intervaloRepeticion = r;
}
public void UpDate(Observado o ){
        Reloj rel = (Reloj)o;
        if(rel.getHora() = horaDeAlarma)
                this. activar();
}
public void activar(){ }
public void desactivar(){ }

}
```

d. (0.5) Implemente en su lenguaje de programación favorito un método dentro de una clase cualquiera, la configuración y puesta en marcha de un reloj con segundero y un minutero, fijando la hora en 00:00, con la activación de tres alarmas a las horas {25:25, 40:05 y 50:10} todas con una frecuencia de 800Hz y un intervalo de repetición de 3 minutos.

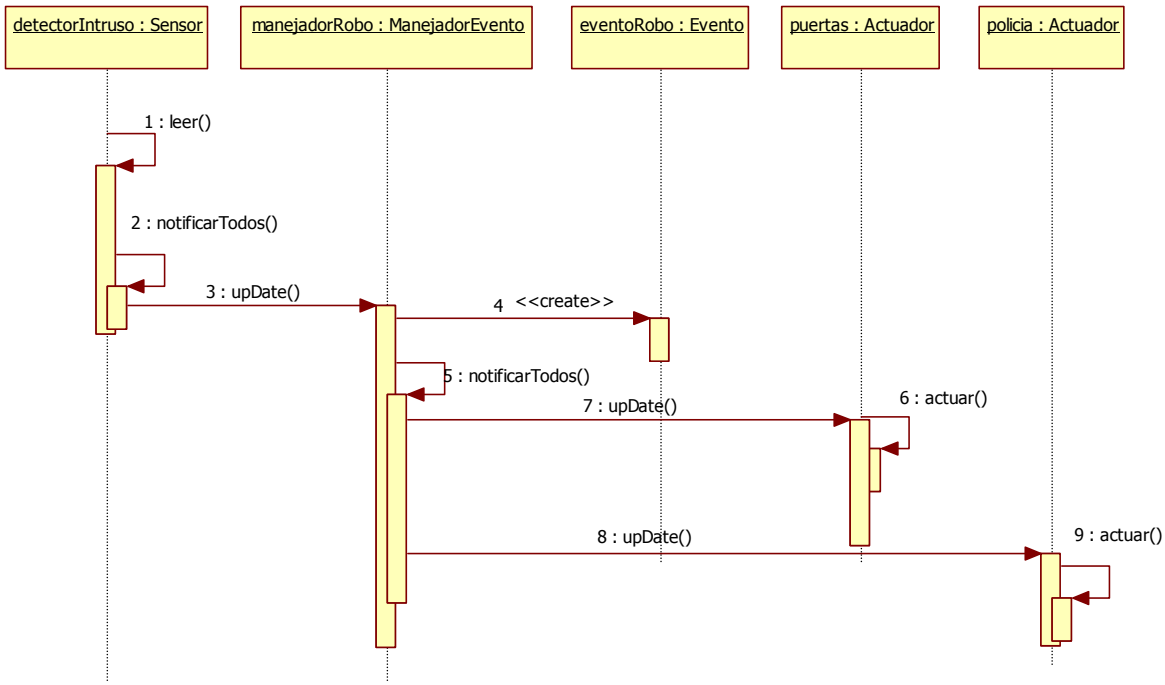
```
public class Cualquiera{
        public void iniciarRelo(){

        Manecilla m = new Manecilla(0, 60, 0, null)
        Manecilla s = new Manecilla(0, 60, 0, m);
        Reloj relojito = new Reloj(s);

        Alarm aLaventarse = new Alarma (“25:25”);
        Alarma aDormir= new Alarma(“40:05”);
        Alarma nueva = new Alarma(“50:10”);
        Display d= new Display();
        relojito. addObsercador(d);
        relojito. addObsercador(aLevantarse);
        relojito. addObsercador(aDormir);
        relojito. addObsercador(anueva);    }
```

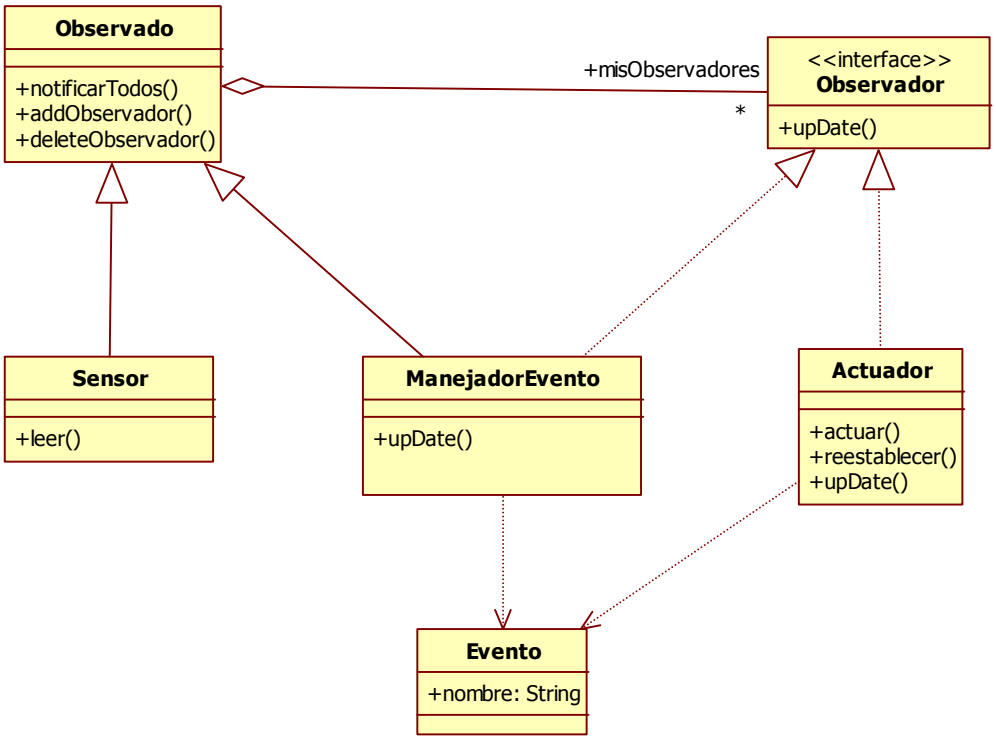
}

Parte B. Caso de Desarrollo 2

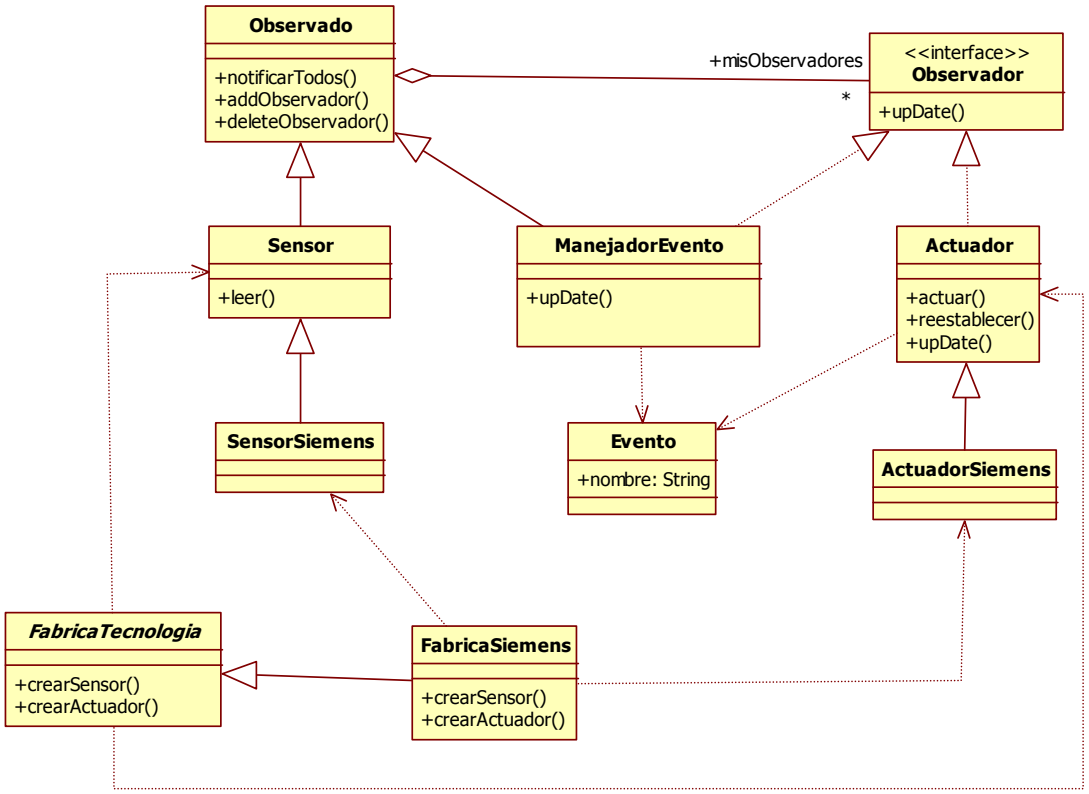
Se está construyendo un sistema para mejorar la seguridad de las viviendas. El sistema usa sensores de humedad, humo, robo, etc, para determinar la presencia de algún evento alarmante. De ocurrir un evento alarmante y dependiendo del tipo de evento, se activan alarmas de robo, incendio, inundación, etc. Además se siguen una serie activaciones que son importantes, por ejemplo si el evento alarmante es de robo, se cierran las puertas, no hay activación de alarmas (para no alarmar al ladrón) y se llama a la policía silenciosamente. Si es incendio se activan las alarmas, se abren las puertas y se llama a la policía.

1. (1.0) Realice el caso de uso "atender un evento de robo" a nivel de diseño.



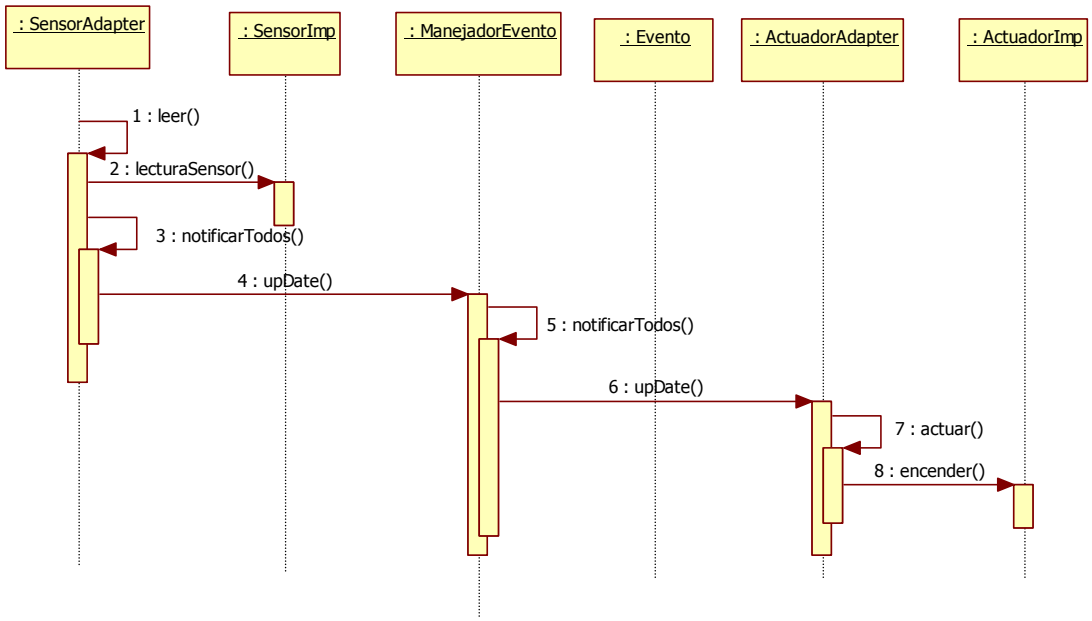


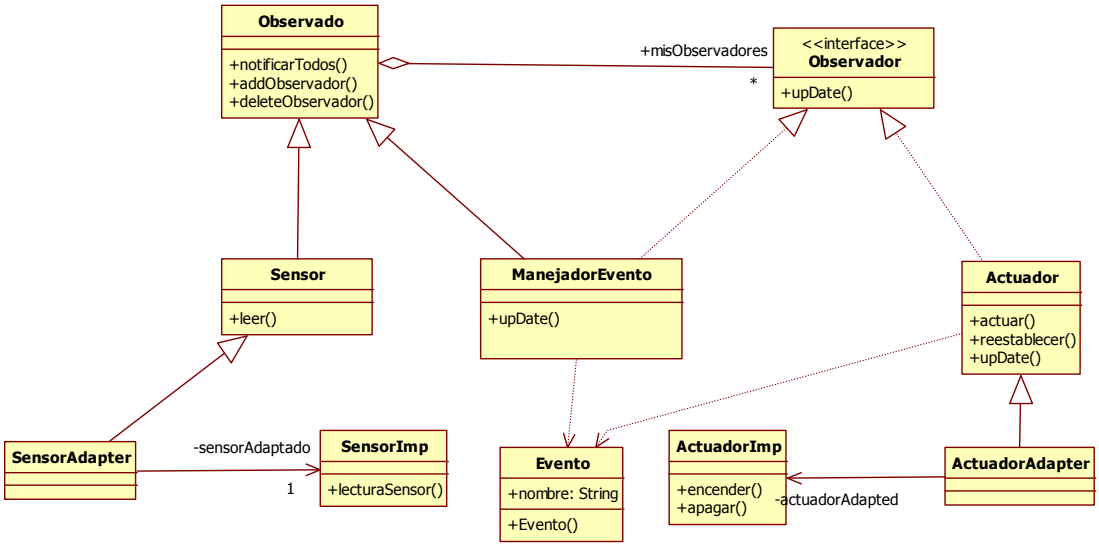
2. (1.0) Extienda el diseño para ganar flexibilidad en alguno de estos cuatro casos(seleccione 1 de los cuatro):
    - El sistema de alarmas se quiere mantener independiente de la tecnología de implementación de Sensores y Actuadores. Para ello no se quiere que la solución requiera conocer las clases específicas al crear los objetos de éstas clases. Justifique su nuevo diseño. Incluya el diagrama de clases y de secuencia actualizados.



De acuerdo al diseño anterior, solo se requieren fábricas de sensores y actuadores de acuerdo a las especificaciones técnicas de cada uno de los fabricantes. Dado que hay varios tipos (no identificables por completo) de sensores y de actuadores para cada fabricante, es importante considerar el paso de un objeto de configuración, o que el fabricante lea un archivo de configuración para cada específico tipo de elemento. El diagrama de secuencia de la solución anterior se mantiene intacto, pues la fabricación busca que el diseño se mantenga independiente de elementos tecnológicos. Aquí se ha continuado utilizando el principio de diseño programe para una interface y no para una implementación.

- Para la implementación del sistema de alarmas se cuenta con el código objeto (el bytecode de java, pero sin el código fuente) tanto de los sensores como de las alarmas, así que se quiere reutilizar dichas implementaciones por tener código de bajo nivel difícil de crear. Incluya el diagrama de clases y el diagrama de secuencia actualizados.

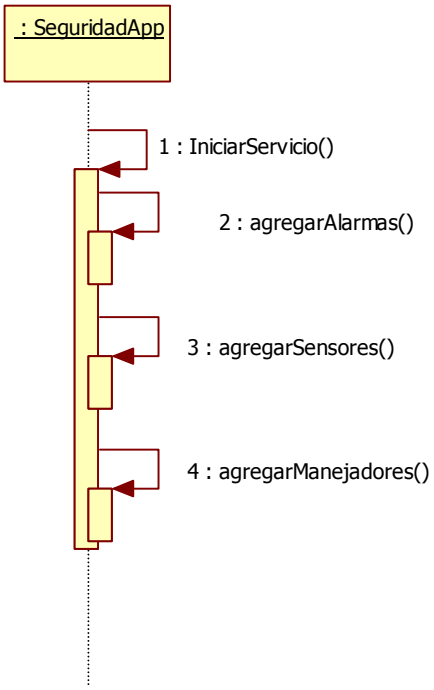




Del diseño anterior se puede observar que dos clases con la mismas interfaces de sensor/actuador han sido usadas para envolver (wrapper) las clases heredadas (bytecode). Aquí se favorece el principio favorezca la herencia a la implementación. Las clases envolventes cumplen la interface que requiere el resto del sistema e internamente llaman a los métodos heredados en lugar de implementarlos (reutilización de caja negra).

- Para la implementación del sistema de alarmas, se desea implementar un mini-framework en el que sea más fácil armar y poner en ejecución el sistema de sensores y alarmas por parte del desarrollador de una solución específica. Incluya el diagrama de clases y de secuencia actualizado.

El diagrama de secuencia es el mismo, lo único sería considerar el armazón de la aplicación:



El esquema del observador y del observable permiten que la aplicación tenga la abstracción de un framework, sólo queda permitirle al programador de una aplicación específica ocultarle la organización a través de un esquema de inicialización reutilizable, sin embargo dejándole la oportunidad de hacer cada paso a su gusto. Para ello se ha utilizado el principio de Hollywood o de control invertido. Nosotros llamamos el código nuevo, a nivel de framework es abstracto, con el fin de que el motor de ejecución sea reutilizable en muchos escenarios en los que sensores, manejadores de evento y actuadores se vayan a coordinar.

- Para la implementación del sistema de alarmas, se tienen algunas alarmas ubicadas en otras máquinas, para ello se requiere contar con elementos que los representen en nuestro sistema local de alarmas para mantener el diseño, pero que puedan acceder de forma remota a los actuadores reales.

Es una solución similar a la del adaptador, sólo que no se adapta el servicio sino que se alcanza un servicio remoto. El adapter se convierte en el proxy y la clase bycode se convierte en la clase del objeto remoto.

3. (0.5) Programe en su lenguaje favorito dos elementos claves (dos clases, de las nuevas obtenidas en el punto 2) definidos en el nuevo diseño obtenido en el punto 2.

Implementaremos algo de cada caso

```
public abstract class FabricaTecnologia {

        public abstract Sensor crearSensor(params);

        public abstract Actuador crearActuador(params);

}


public  class FabricaSiemens extends FabricaTecnologia{

        public Sensor crearSensor(params){ // aqui va la implementación}

        public Actuador crearActuador(params){//aquí va la implementación};

}
```

```
public class ActuadorAdapter extends Actuador{

prívate ActuadorImp actuadorAdaptado;

public void actuar(){

        actuadorAdaptado.encender();

}

}
```

```
public abstract class SeguridadApp{

public IniciarServicio{
```

```
            agregarSensores(),
            agregarActuadores();
            agregarManejadores();
    }
    public abstract void agregarSensores();
    public abstract void agregarActuadores();
    public abstract void agregarManejadores();

    }
```

¡Que le vaya bien!