Obligatorio

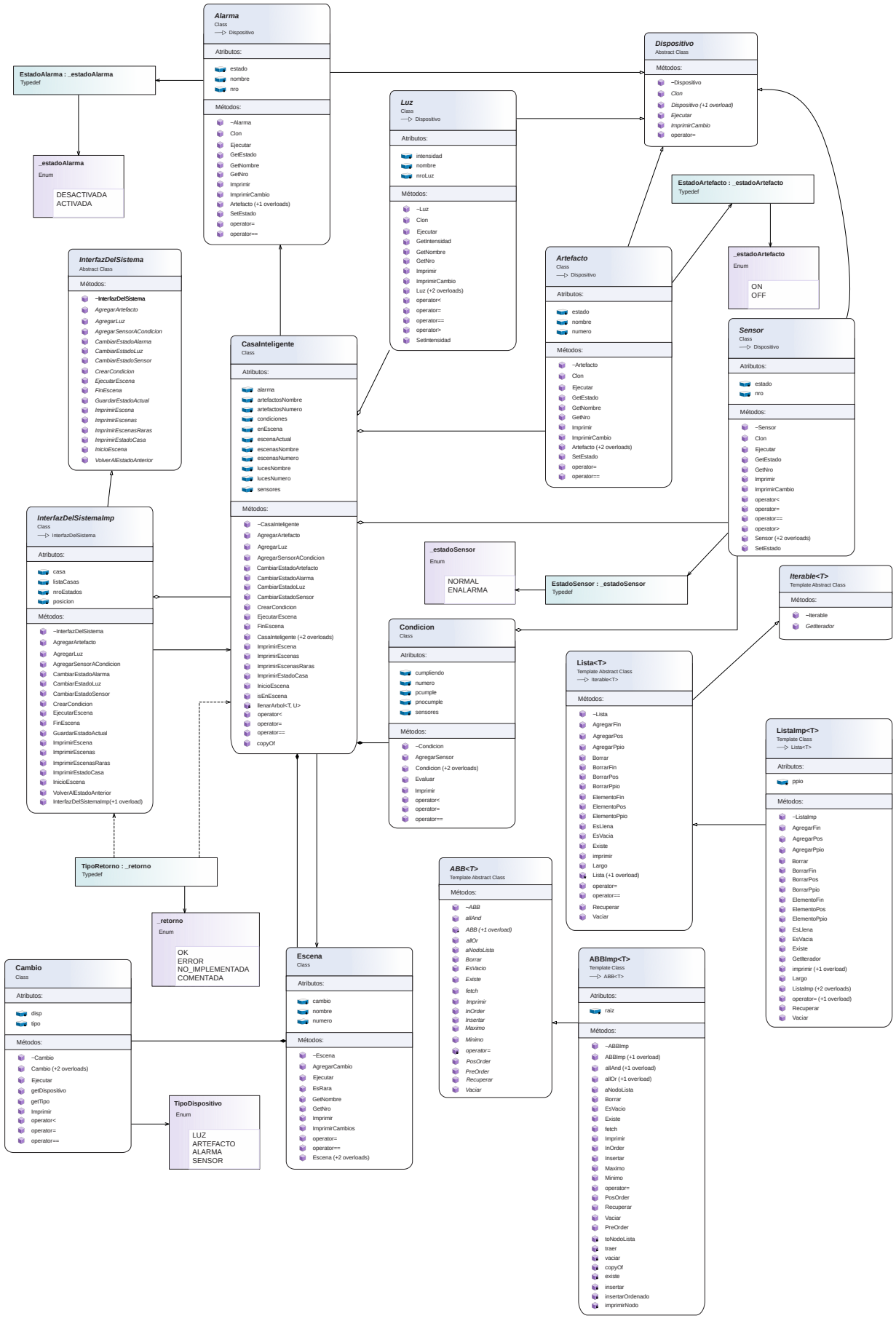
ESTRUCTURAS DE DATOS Y ALGORITMOS 1

Matías Gutierrez

(200244)

Contenido

1. Análisis y diseño del sistema.



2. Implementaciones escogidas.

TAD's Escogidos:

Los TAD's que escogimos como estructuras de datos fueron ABB y Lista; con sus implementaciones correspondientes siendo ABBImp (Árbol Binario de Búsqueda) y ListaImp (Lista simplemente encadenada) respectivamente.

Elegimos esas implementaciones porque fueron las que creemos que el curso le dio mayor enfoque (por lo que tenemos más practica en utilizarlas ante cualquier otra) además de ser potentes y eficientes herramientas que nos asistieron a resolver los problemas planteados por la letra. Por ejemplo en ciertos métodos, era necesario que en promedio fueran realizados en $O(log_2 n)$ siendo n la cantidad de objetos preexistentes, por lo que sí o sí necesitabamos un ABB o un array utilizando la búsqueda por bipartición.

Dos funciones con algoritmos más interesantes a nuestro criterio:

- Funciones de escena:

IniciarEscena() y EjecutarEscena() a nuestro parecer fueron de las más interesantes de realizar porque teníamos que de alguna manera, al iniciarse una escena, guardar los cambios a dispositivos en la lista de cambios de la escena y luego aplicarlos a la casa.

Nuestra forma de enfrentarnos a esta tarea fue primero analizar de que forma estaban relacionados esos dispositivos, cuya respuesta es que claramente todos son implementaciones del TAD Dispositivo.

Cada una de las escenas tiene una lista de cambios como mencionamos anteriormente, y el constructor de cambio recibe un dispositivo, que puede ser o Sensor, o Alarma o Luz, entonces al recibirlo, utilizando typeid.name() averiguamos de que clase es, y seteamos un enum que creamos con el nombre de la clase, para tenerlo identificado.

Al saber de qué tipo es cada uno de los dispositivos dentro de cada cambio, al ejecutarlo, casteando al tipo de dispositivo que es (luz, sensor o alarma) con la instrucción dynamic_cast obtuvimos acceso a los métodos públicos de cada uno de ellos y realizamos el respectivo cambio de estado con los valores guardados en el dispositivo que se encontraba dentro del cambio.

Estas operaciones funcionan porque estamos indicándole a C++ que sabemos cuál es la implementación que estamos usando en cada caso, si estuviéramos equivocados, se caería el programa (exception).

- Saber si una escena es rara:

Otro algoritmo que nos parece interesante destacar es la funcion de escena esRara() que indica true si han habido dos o mas cambios de alarma, dos o mas cambios en la misma luz o en el mismo artefacto.

Para realizar eficientemente la búsqueda sabíamos que tendríamos que recorrer la lista de cambios la menor cantidad de veces posibles y luego de analizarlo, dimos con una alternativa que nos permite recorrer la lista una sola vez con algún agregado que explicaremos a continuación.

Utilizamos contadores, para la alarma era muy fácil porque es una sola, entonces con un simple int fue suficiente, pero en el caso de las luces y los artefactos era otra historia; utilizamos el map<> de C++.

Elegimos utilizarlo porque sabemos que al ser nativo del lenguaje su eficiencia es incuestionable y nos fue bastante facil de implementar. Creamos dos maps `map<Cadena, int> luces` y `map<Cadena, int> artefactos`.

A medida que recorremos la lista de cambios con el iterador, si es una alarma sumamos 1 a su contador, y si es una luz o un artefacto, mapeamos el nombre (que sabemos que es único, actuando como key) con otro contador que será propio de cada uno en particular. De esta manera, si se hacen cambios en la misma luz o artefacto, se le sumará 1 al contador asociado con su nombre.

Luego de esto fue muy sencillo, si el contador de la alarma era mayor o igual a 2, retornamos true y utilizando dos for (no anidados) recorrimos cada uno de los maps, y si existía alguno de los nombres de luces o artefactos asociado con un entero mayor o igual a dos, también retornamos true.
De esta manera resolvimos el problema sin estructuras anidadas.

3. Memoria de las tareas realizadas.

Obligatorio realizado en conjunto

Memoria de Tareas Realizadas					
Operación	*Fue Implementada*	*Estado de entrega*	*Desarrollada por:*		*Comentarios*
			Bruno Vezoli	*Matías Gutierrez*	
InterfazDelSistema	SI	4. ejecuta sin errores	X	X	
~InterfazDelSistema	SI	4. ejecuta sin errores	X	X	
AgregarLuz	SI	4. ejecuta sin errores	X	X	
AgregarArtefacto	SI	4. ejecuta sin errores	X	X	
CambiarEstadoLuz	SI	4. ejecuta sin errores	X	X	
CambiarEstadoArtefacto	SI	4. ejecuta sin errores	X	X	
CambiarEstadoAlarma	SI	4. ejecuta sin errores	X	X	
ImprimirEstadoCasa	SI	4. ejecuta sin errores	X	X	
CrearCondicion	SI	4. ejecuta sin errores	X	X	
AgregarSensorACondicion	SI	4. ejecuta sin errores	X	X	
CambiarEstadoSensor	SI	4. ejecuta sin errores	X	X	
InicioEscena	SI	4. ejecuta sin errores	X	X	
FinEscena	SI	4. ejecuta sin errores	X	X	
EjecutarEscena	SI	4. ejecuta sin errores	X	X	
ImprimirEscenas	SI	4. ejecuta sin errores	X	X	
ImprimirEscena	SI	4. ejecuta sin errores	X	X	
ImprimirEscenasRaras	SI	4. ejecuta sin errores	X	X	
GuardarEstadoActual	SI	4. ejecuta sin errores	X	X	
VolverAlEstadoAnterior	SI	4. ejecuta sin errores	X	X	

4. Conclusiones.

No tenemos funcionalidades no implementadas y creemos que todas funcionan correctamente.