



SISTEMA COMPLEJO DE EMERGENCIAS (SCE)

Captura y Representación de Decisiones de Diseño

DESCRIPCIÓN BREVE

Como equipo de Arquitectos Software nos encontramos ante un problema de diseño, donde deberemos encontrar los requisitos del problema y diseñar la arquitectura Software más conveniente para resolverlo.

Javier Barrio, María Gutiérrez, David Robles, Álvaro Nogueras, Adrián Gómez de Juan y Alex Aguilar.

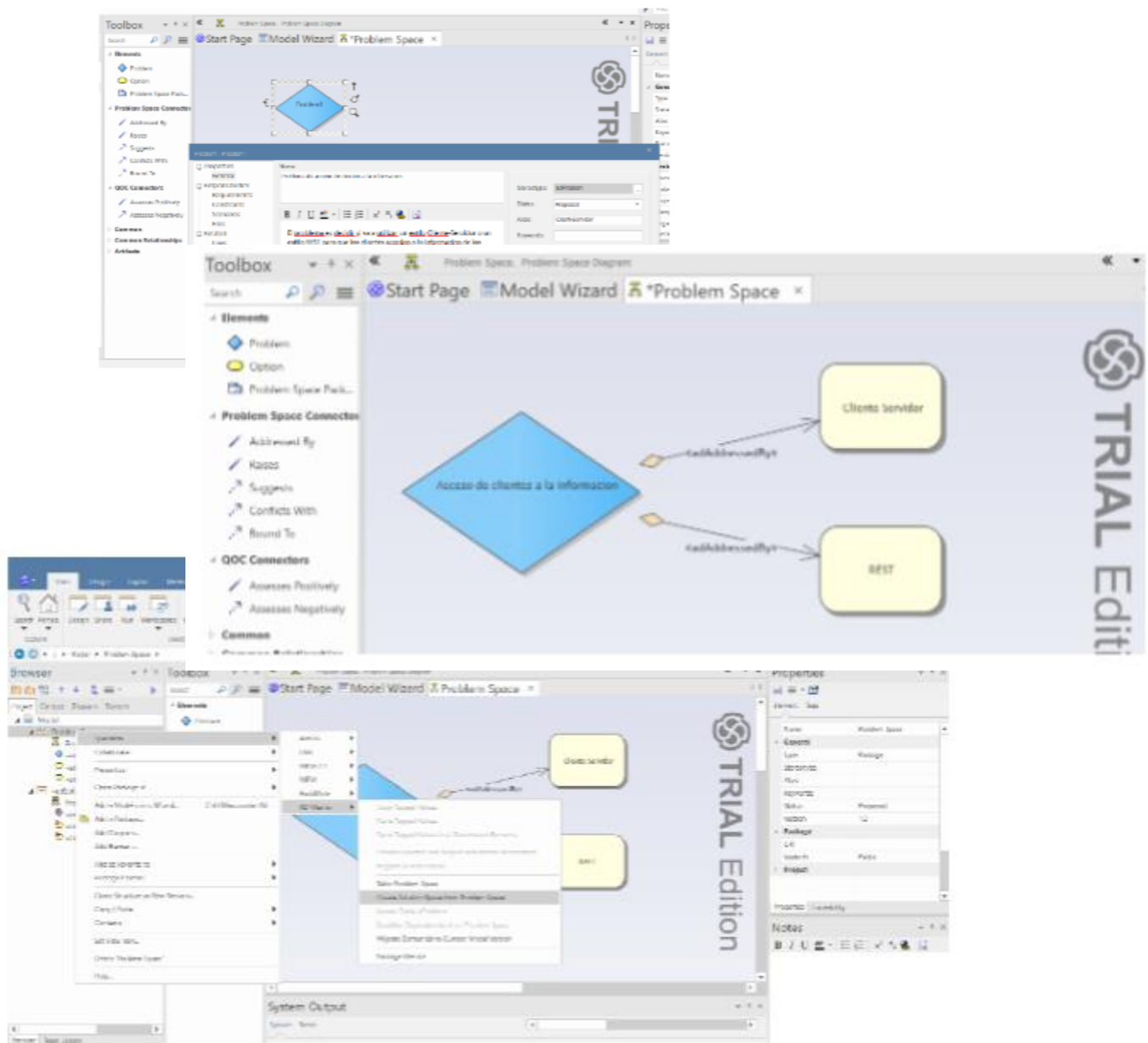
Diseño y Arquitectura de Software

Práctica 1: Captura y Representación de Decisiones de Diseño

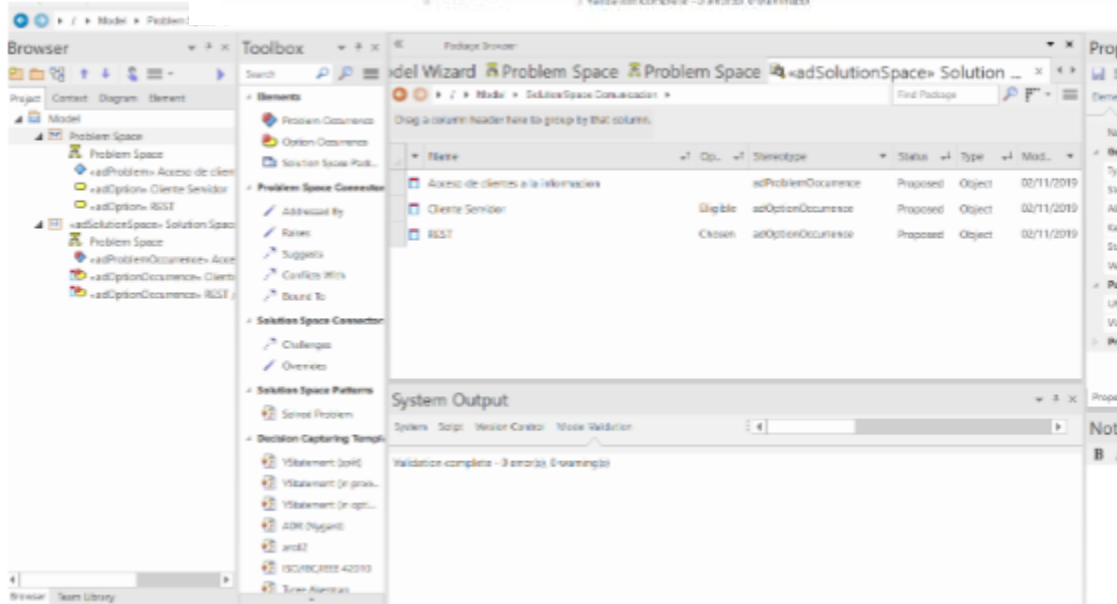
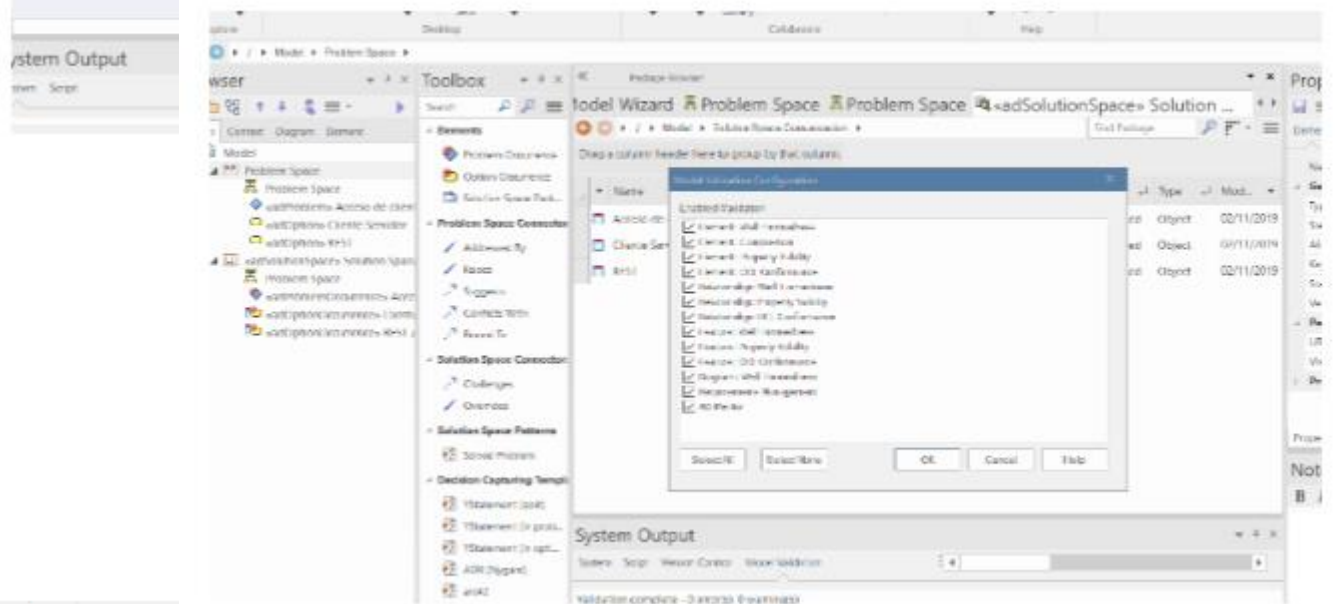
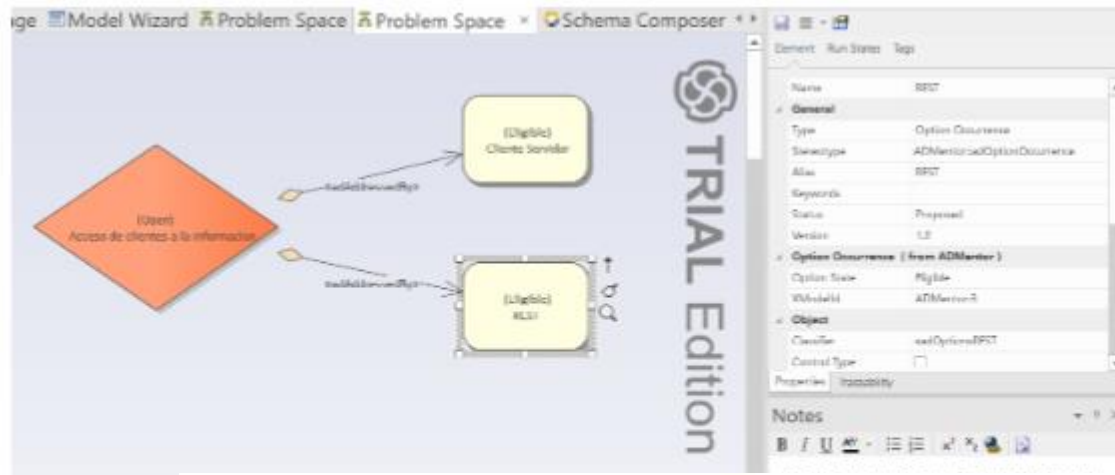
1. Roles:

- Arquitectos de Software Seniors: Javier Barrio y María Gutiérrez.
- Arquitectos Software Juniors: Adrián Gómez De Juan y Alejandro Aguilar.
- Arquitectos Software Cognitivos: David Robles y Álvaro Noguerales.

2. Capturas de Pantalla de ADMentor:



Práctica 1: Captura y Representación de Decisiones de Diseño



Práctica 1: Captura y Representación de Decisiones de Diseño

2. Descripción de los resultados para cada tarea:

Título	Estilo General MVC
ID	D01
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Modificada
Requisitos	RF1: Separar el diseño de la APP en Modelo, Vista y Controlador RF2: Definir como Modelo, la BD y el Gestor del sistema. RF3: Definir como vista a la Interfaz. RF4: Definir como Controlador todas las sub-partes del Gestor.
Decisiones Alternativas	Se podría modificar la estructura del patrón con más capas si fuese necesario o utilizar un estilo similar como el Modelo Vista Presentador.
Resultado de la decisión	Hemos decidido usar el estilo MVC de manera general porque nos permite separar los componentes de nuestra aplicación dependiendo de la responsabilidad que tienen, esto nos permite cambiar parte del código sin afectar al resto.
Pros	La posibilidad de reutilizar código y la separación de conceptos.
Contras	Habría que tener en cuenta que el estilo MVC es un estilo orientado a objetos, por lo que, el lenguaje a la hora de desarrollarlo también deberá serlo
Unión con otra decisión	
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Estilo Capas Gestor
ID	D02
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Rechazada
Requerimientos	RF5: Tratar cada parte gestionada por el Gestor como una Capa
Decisiones Alternativas	Aprovechar el propio modelo MVC
Resultado de la decisión	Decidimos el uso del estilo por capas para tener cierta autonomía entre las partes del gestor manteniendo la conexión entre ellas.
Pros	Facilita la estandarización, eliminando dependencias y conteniendo cambios a una determinada capa.
Contras	Perdida de la eficiencia, redundancia entre capas y una dificultad de diseño correcta entre capas.
Unión con otra decisión	D01
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Estilo Cliente Servidor Usuarios
ID	D03
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	<p>RF6: Necesidad de un servidor para almacenar datos, en nuestro caso, usaríamos la BD.</p> <p>RF7: Necesidad de una API REST para la conexión entre cliente y servidor.</p>
Decisiones Alternativas	Como alternativa se podría implementar una arquitectura Cliente-Cola-Cliente en la que el servidor actúa como cola que va almacenando las peticiones.
Resultado de la decisión	Decidimos la utilización del estilo cliente-servidor para la comunicación con los clientes para poder notificarles los eventos correspondientes gestionados por el gestor.
Pros	Los clientes pueden acceder a los datos mediante un servidor.
Contras	Necesidad de conexión a Internet.
Unión con otra decisión	D01
Unión con Arquitectura	Arquitectura Cliente-Servidor

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Estilo Eventos Gestor
ID	D04
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	RF8: Conexión a Internet con una alta velocidad de datos.
Decisiones Alternativas	Como alternativa se podría implementar el patrón observer en los objetos generados así poder notificar los cambios.
Resultado de la decisión	Decidimos la utilización del estilo por eventos debido a la necesidad de controlar las llamadas, comunicaciones, emergencias y eventos; todos ellos funcionando mediante notificaciones.
Pros	El estilo por eventos es bastante simple, además nos permite el uso de una modularidad para los diferentes eventos a comunicar. La entrega de los eventos a tiempo real y la desvinculación entre productores y consumidores.
Contras	Necesidad de conexión a Internet, posibilidad de desborde, no existe garantía de respuesta por parte del suscriptor
Unión con otra decisión	D01-D03
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Patrón Singleton / Builder Gestor
ID	D05
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	Utilizar la exclusión mutua en el método de creación de la clase del patrón, para evitar crear dos instancias al mismo tiempo.
Decisiones Alternativas	Creación de dependencias entre clases
Resultado de la decisión	Hemos elegido el patrón singleton para definir la creación de una única estancia de la instancia del gestor, además el patrón builder nos permitirá crear instancias de las emergencias, llamadas, empleados, eventos, usuarios, recursos y comunicaciones dentro del gestor.
Pros	No permitimos que por error se creen dos instancias del gestor, y facilidad y similitud en la creación de instancias de los elementos dentro del gestor.
Contras	El patrón singleton podría producir el alto acoplamiento. En el caso del builder debemos tener en cuenta que las clases producidas deben ser mutables y no garantiza la inicialización de los campos de la clase.
Unión con otra decisión	D04
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Patrón Abstract Factory Objetos Gestor
ID	D06
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Rechazada
Requerimientos	No se necesitan requerimientos previos
Decisiones Alternativas	Utilizarlo en el modelo y no en el controlador para distinguir los objetos
Resultado de la decisión	Usaremos el patrón Abstract Factory para evitar la mezcla de objetos dentro del Gestor y que se aprecie a que familia de las distintas partes del Gestor pertenece.
Pros	Brinda flexibilidad al aislar clases concretas
Contras	Para agregar nuevos productos debemos modificar las clases abstractas y concretas.
Unión con otra decisión	D04-D05
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Patrón Factory Method Empleados y Usuarios
ID	D07
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Rechazada
Requerimientos	No se necesitan requerimientos previos
Decisiones Alternativas	Utilizarlo en el modelo y no en el controlador para distinguir los objetos
Resultado de la decisión	Usaremos el patrón Factory Method para ocultar la diversidad de casos particulares que puede tener tanto un empleado como un usuario.
Pros	Brinda flexibilidad en la creación de objetos, además de facilitar futuras ampliaciones en objetos, por otro lado, facilita la jerarquía entre clases paralelas.
Contras	Obligamos al cliente a definir subclases de una clase Creadora
Unión con otra decisión	D04-D05-D06
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Patrón Facade Interfaces del Gestor
ID	D08
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	Una buena definición y diseño del Gestor
Decisiones Alternativas	Podría acceder directamente al subsistema pero eso implicaría una menor eficiencia y la necesidad de que la clase gestor conociese todo el sistema.
Resultado de la decisión	Mediante el patrón Facade proveeremos de una interfaz unificada simple para acceder a cada grupo de interfaces del gestor.
Pros	Reducimos el número de objetos con los que el cliente trata, se reducen las dependencias, las aplicaciones pueden usar clases de los subsistemas si las necesitan.
Contras	En caso de utilizar una fachada global los usuarios podrían utilizar solo una pequeña parte, por lo que deberíamos utilizar fachadas más específicas.
Unión con otra decisión	D04-D05-D06-D07
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Patrón ChainofResponsibility Comunicación
ID	D09
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	No se necesitan requerimientos previos
Decisiones Alternativas	Se puede usar el patrón Composite como alternativa para construir una estructura común entre objetos.
Resultado de la decisión	Usaremos el patrón ChainofResponsibility para las comunicaciones entre objetos de diferentes clases.
Pros	Poseer una estructura común entre objetos reduciendo acoplamiento y asignando flexibilidad en las responsabilidades a objetos.
Contras	Si no configuramos la estructura correctamente una petición puede quedarse sin tratar.
Unión con otra decisión	D04-D05-D06-D07-D08
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Patrón State Empleados
ID	D10
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	Contemplar que el empleado puede estar libre o no
Decisiones Alternativas	Usar el patrón Flyweight para generar objetos que contengan información común y concreta.
Resultado de la decisión	Mediante el patrón State modificamos el comportamiento de los empleados pudiendo especificar si están libres de trabajo o no.
Pros	Brinda flexibilidad al aislar clases concretas
Contras	Existe una extrema complejidad en el código cuando se intentan administrar comportamientos diferentes según el número de estados diferentes. Además, incrementa el número de sub clases.
Unión con otra decisión	D04-D05-D06-D07-D08-D09
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Patrón ObserverNotificación estado de los Empleados
ID	D11
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	Tener bien implementada la D10
Decisiones Alternativas	Como alternativa se podría utilizar el patrón mediator para que los objetos se comuniquen entre sí.
Resultado de la decisión	Notificaremos los cambios en los estados de los empleados mediante el Patrón Observer a los objetos que necesiten conocer estos cambios.
Pros	Abstrae el acoplamiento entre el sujeto y el observador consiguiendo mayor independencia.
Contras	Desconocimiento acerca de los motivos de una actualización.
Unión con otra decisión	D04-D05-D06-D07-D08-D09-D10
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Patrón Publish-Subscribe Notificación Usuarios
ID	D12
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	Contemplar la suscripción de Usuarios
Decisiones Alternativas	Implementar el patrón observer junto con la arquitectura de eventos implementada para notificar a los usuarios.
Resultado de la decisión	Notificaremos los cambios no programados mediante eventos notificables a los usuarios suscritos.
Pros	Los editores no necesitan saber la existencia de los usuarios suscritos. Mejora el estilo cliente-servidor tradicional permitiendo las operaciones paralelas, y el almacenamiento en caché de mensajes.
Contras	Se debe implementar cuidadosamente para evitar pérdidas de mensajes asegurando la entrega. También se pueden presentar retrasos dependiendo del número de aplicaciones que usan el sistema, y el volumen de mensajes.
Unión con otra decisión	D04-D05-D06-D07-D08-D09-D10-D11
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Modificación D01
ID	D13
Date	16/10/2019
Creadores	David Robles, Álvaro Noguerales
Estado	Aceptada
Requisitos	Contemplar la D01
Decisiones Alternativas	Mantener la clase recursos en el Gestor y creación de un registro de distintos usuarios en el sistema.
Resultado de la decisión	<p>Partiendo de la D01, en la que habíamos contemplado la existencia de una clase Recursos dentro del Gestor, trasladamos dicha clase fuera del Gestor y funcionando en paralelo con él, siendo parte ahora del Modelo.</p> <p>También eliminaremos la clase Usuario que se había ubicado dentro del Gestor, ya que no se especifica que los Usuarios puedan no tener acceso a todo el sistema.</p>
Pros	El Gestor se hará cargo a la gestión de eventos y peticiones de usuarios únicamente.
Contras	Perdida de la gestión de usuarios.
Unión con otra decisión	D01
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Modificación D06 y D07
ID	D14
Date	16/10/2019
Creadores	David Robles, Álvaro Nogueras
Estado	Aceptada
Requerimientos	Utilizar en patón Facade implementado en la decisión D08 para que el sistema no tenga por qué conocer ese subsistema.
Decisiones Alternativas	No se contemplan
Resultado de la decisión	Trasladamos el uso de ambos patrones al Modelo para poder ordenar y distinguir los diferentes objetos
Pros	Toda la información requerida por el sistema se encontrará en el Modelo.
Contras	Se deben implementar adecuadamente ya que en esta parte del sistema es donde conviven la mayor parte de los recursos.
Unión con otra decisión	D06-D07
Unión con Arquitectura	Arquitectura M-V-C

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Modificación D13
ID	D15
Date	22/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Modificada
Requerimientos	Una nueva BD para el almacenamiento de usuarios y un servidor para su uso que admita un sistema almacenamiento caché
Decisiones Alternativas	Se podría utilizar una arquitectura SOAP(Simple Object Access Protocol) pero esta sería más pesada y más compleja de implementar.
Resultado de la decisión	Para la conexión de la aplicación con los usuarios vamos a utilizar un estilo cliente-servidor a través de una API REST
Pros	Crear un sistema independiente entre cliente y servidor, necesitando el estilo REST menos recursos del servidor que otras opciones. Por último, mencionar que la API REST permite independencia en el lenguaje entre el cliente y el servidor
Contras	Mayores tiempos de desarrollo, ya que, tienes que montar un sistema API, mayor rigidez en el desarrollo debido a que pueden producirse situaciones de desincronización.
Unión con otra decisión	D13
Unión con Arquitectura	

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Modificación D13
ID	D16
Date	22/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Rechazada
Requerimientos	La situación planteada en la decisión 13
Decisiones Alternativas	Como decisión alternativa, los recursos se trasladarían dentro del gestor conectados únicamente a las emergencias, ya que, estos recursos realmente son únicamente necesarios para ellas.
Resultado de la decisión	Los recursos deberán tener una conexión directa con la BD SCE para disminuir retardos al no tener que pasar por el gestor.
Pros	Disminuimos el tiempo de espera al actualizar el estado de los recursos y evitamos clases intermedias.
Contras	Mayor infraestructura en el diseño
Unión con otra decisión	D13
Unión con Arquitectura	

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	Modificación D15 y D16
ID	D17
Date	23/10/2019
Creadores	David Robles, Álvaro Noguerales
Estado	Aceptada
Requerimientos	Lo contemplado en la D15
Decisiones Alternativas	Almacenar los recursos dentro del sistema y que estos se actualicen de manera periódica.
Resultado de la decisión	Los recursos finalmente estarán almacenados en un servidor y conectados únicamente a las emergencias, a través del gestor, un cliente enviará la petición al servidor para usar los recursos. Además, cada petición del cliente debería poder ser cacheable por un token que identifique la petición y las tareas que realiza.
Pros	Crear un sistema independiente entre cliente y servidor, necesitando el estilo REST menos recursos del servidor que otras opciones. Por último, mencionar que la API REST permite independencia en el lenguaje entre el cliente y el servidor
Contras	Mayores tiempos de desarrollo, ya que, tienes que montar un sistema API, mayor rigidez en el desarrollo debido a que pueden producirse situaciones de desincronización.
Unión con otra decisión	D13
Unión con Arquitectura	

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	API Decode
ID	D18
Date	30/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	Una nueva BD para el almacenamiento y un servidor para su uso que admita un sistema almacenamiento caché
Decisiones Alternativas	
Resultado de la decisión	Hacer uso de una API para la traducción simultánea en tiempo real.
Pros	Rapidez, reducción de costes y calidad en la traducción.
Contras	Capacidad lingüística limitada, traducciones fuera de contexto.
Unión con otra decisión	D15
Unión con Arquitectura	

Práctica 1: Captura y Representación de Decisiones de Diseño

Título	API Traceroute
ID	D9
Date	15/10/2019
Creadores	Javier Barrio, María Gutiérrez
Estado	Aceptada
Requerimientos	Una nueva BD para el almacenamiento y un servidor para su uso que admita un sistema almacenamiento caché
Decisiones Alternativas	No se contemplan
Resultado de la decisión	Hacer uso de una API para la optimización de rutas en casos de emergencia.
Pros	Rapidez. fácil de usar y puede mostrar rutas alternativas en casos de problemas ajenos como el tráfico.
Contras	Puede ser costoso y en caso de falta de actualización tener problemas respecto a tráfico u obras
Unión con otra decisión	D15
Unión con Arquitectura	

3. Tabla de Tiempos:

Semana	Iteración	Time in ADD (AS)	Reflection Time (ASS-ASC)	Time in refined ADD (ASS)	Design ADD Time (ASJ)
2	1	90'	60'	30'	153'
3	1	40'	30'	45'	125'
4	1	60'	45'	30'	120'

Práctica 1: Captura y Representación de Decisiones de Diseño

4. Conclusiones en base a lecciones aprendidas:

Como ASS hemos podido profundizar en la captura de decisiones, permitiéndonos aumentar nuestros conocimientos en estos aspectos. Por otro lado, mejorar la comunicación con los ASC a través de conversaciones en las que debatíamos las decisiones tomadas.

Para finalizar, mencionar los numerosos patrones y estilos vistos en la búsqueda de las diferentes opciones posibles a nuestra Arquitectura.

Como ASC hemos aprendido a valorar los posibles riesgos y alternativas de cada decisión e intentar encontrar así la mejor arquitectura posible para el sistema. Para ello hemos tenido que investigar y profundizar en algunos conceptos de la arquitectura software desconocidos hasta ahora para nosotros.

Otro aspecto aprendido ha sido como varias arquitecturas pueden convivir dentro de un sistema para llegar a las soluciones de los problemas que el software pretende resolver.

Por otra parte, hemos mejorado a la hora de comparar ideas y debatir decisiones de diseño, lo que se ha reflejado en una mejor comunicación con los ASS.

Como ASJ nos hemos dado cuenta de que realizar una captura de decisión de diseño es una tarea que requiere de varias iteraciones en las que, para ir construyéndola de forma iterativa, se han de tener en cuenta los Pros y los Contrás. Es una tarea que requiere un mínimo de tiempo y dedicación, así como la coordinación de los distintos miembros que componen el equipo o las personas que están capturando las decisiones de diseño.

Es una inversión que, desde el punto de vista empresarial, ahorra muchos costes ya que con el paso del tiempo y el transcurso de la gente las ideas que se tenían claras en un principio dejan de serlo.

5. Bibliografía:

- Servicio de Informática ASP.NET MVC: " <https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html>".
- J. de Andalucía (MVC): " <http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/122>".
- Capdevila, Albert: " <https://albertcapdevila.net/patron-builder-csharp-net>"
- Wikipedia: " https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas"
- Redespomactividad: " <https://redespomactividad.weebly.com/modelo-cliente-servidor.html>"
- Microsoft Azure: " <https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/event-driven>"
- Eljaviador: " <http://eljaviador.com/el-objeto-unico-patron-singleton.html>"
- Departamento de ciencias de la computación UdG: " <https://elvex.ugr.es/decsai/information-systems/slides/52%20Middleware%20-%20Publish-Subscribe.pdf>"
- Sánchez, Giovanni: " <http://giovanni-sanchez.blogspot.com/2009/05/chain-of-responsibility.html>"
- Libro Patrones de Diseño, Erich Gamma.
- Diagramas UML: " <https://diagramasuml.com/>"
- API REST: " <https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos>"

Práctica 1: Captura y Representación de Decisiones de Diseño

6. Anexo con todos los tiempos Estimados:

Semana	Iteración	Time in ADD (AS)	Reflection Time (ASS-ASC)	Time in refined ADD (ASS)	Design ADD Time (ASJ)
2	1	60' (+30)	30' (+30)	45' (-15)	120' (+33)
3	1	60' (-20)	45' (-15)	45' (0)	150' (-25)
4	1	30' (+30)	45' (0)	45' (-15)	90' (+30)

*(Entre paréntesis se muestra la diferencia de tiempo entre el que estimamos, y el tiempo real)

7. Documentos en GitHub:

Enlace GITHUB: (Público) <https://github.com/adriang5/PracticaDAS>

Branch: Semana2 ▾ New pull request

Create new file Upload files Find file Clone or download ▾

This branch is 15 commits ahead, 4 commits behind master. [Pull request](#) [Compare](#)

JaviBarrio6 Actualizacion Latest commit ebdceec 13 seconds ago

Decisiones de Diseño	Actualizacion	13 seconds ago
UMLs	Actualizacion	13 seconds ago
DAS-P1-MariaGutierrez.docx	Actualizacion	13 seconds ago
Enunciado Práctica.pdf	Actualizacion	13 seconds ago
README.md	Primer DASComentario	last month
Requisitos.pdf	Nuevas Decisiones	3 days ago
~\$S-P1-MariaGutierrez.docx	Actualizacion	13 seconds ago

Branch: Semana3 ▾ New pull request

Create new file Upload files Find file Clone or download ▾

This branch is 9 commits ahead, 4 commits behind master. [Pull request](#) [Compare](#)

JaviBarrio6 Merge branch "Semana3" of <https://github.com/adriang5/PracticaDAS> int... Latest commit 884ccb8 5 minutes ago

Decisiones de Diseño	Inicio Semana 3	yesterday
UMLs	Estilo Cliente Servidor	23 hours ago
DAS-P1-MariaGutierrez.docx	Actualizacion .doc	5 minutes ago
DAS-P1-MariaGutierrez.pdf	Inicio Semana 3	yesterday
Enunciado Práctica.pdf	Inicio Semana 3	yesterday
README.md	Primer DASComentario	last month
Requisitos.pdf	Inicio Semana 3	yesterday

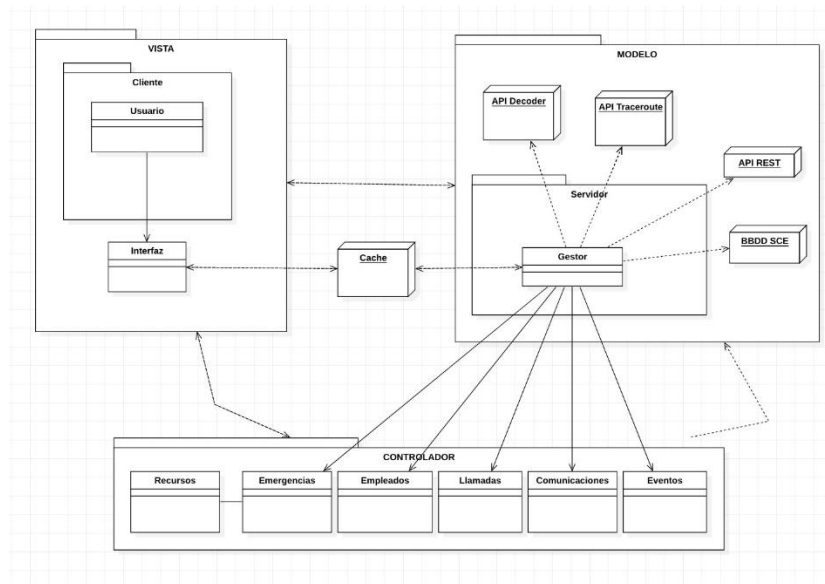
Práctica 1: Captura y Representación de Decisiones de Diseño

8. Arquitecturas producidas en cada Iteración:

El estilo general de nuestra App será estilo M-V-C (Modelo Vista Controlador): Modelo-vista-controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

Dada nuestra distribución y el estilo M-V-C encontraremos, en el modelo, la BD, el gestor, una API REST para usuarios, una API para rutas y otra API para las traducciones.



El Modelo es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tantas consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del controlador. La API DECODER nos permitirá una traducción textual simultánea en tiempo real para emergencias que afectan a países colindantes o cercanos, mientras que, en segundo lugar, hemos tenido en cuenta hacer uso de otra API, en concreto API TRACEROUTE para optimizar las rutas en casos de emergencia.

El controlador nos lo encontraremos en el trabajo de gestión del gestor.

El Controlador responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un

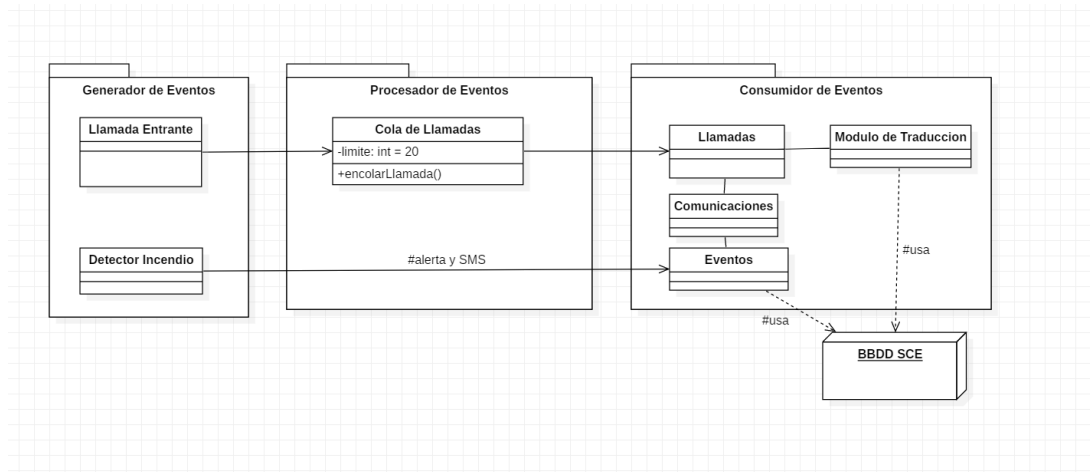
Práctica 1: Captura y Representación de Decisiones de Diseño

documento o por los diferentes registros de una base de datos), por tanto, se podría decir que el ‘controlador’ hace de intermediario entre la ‘vista’ y el ‘modelo’.

En cuanto a la vista, nos encontraremos con una interfaz centralizada.

La vista Presenta el ‘modelo’ (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho ‘modelo’ la información que debe representar como salida. También debemos tener en cuenta, un estilo cliente – servidor para conectar los usuarios con el sistema. La particularidad de este estilo es que la parte del cliente se gestionará desde la interfaz mediante peticiones a una API REST ubicada en el modelo (Servidor) y almacenando datos en caché.

En cuanto al Gestor, utilizaremos un estilo por eventos para su control.



Una arquitectura basada en eventos consta de productores de eventos que generan un flujo de eventos y consumidores de eventos que escuchan los eventos.

Los eventos se entregan casi en tiempo real, de modo que los consumidores pueden responder inmediatamente a los eventos cuando se producen. Los productores se desconectan de los consumidores y los consumidores se desconectan entre sí.

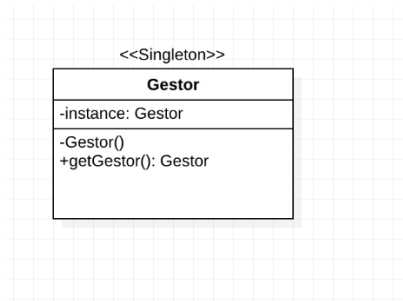
En nuestro caso es apropiado hacer uso de esta arquitectura ya que podemos englobar aquí múltiples requisitos como que el gestor regula los eventos de manera simultánea, el gestor de eventos enviará un SMS y una alerta al sistema.

Además, tendremos los recursos conectados únicamente a las emergencias, ya que, solo son utilizados en ellas. Para interactuar con estas los usuarios lo harán mediante peticiones a través de una API REST.

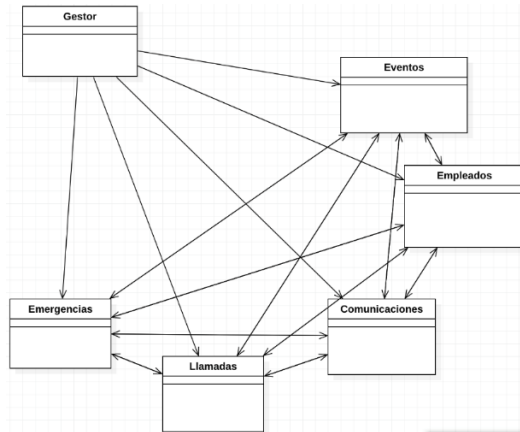
Una vez definidos los estilos, debemos definir los patrones:

Mediante el patrón Singleton nos encargaremos de crear una única instancia del gestor, a la hora de crear emergencias, llamadas, empleador, eventos y comunicaciones usaremos el patrón builder.

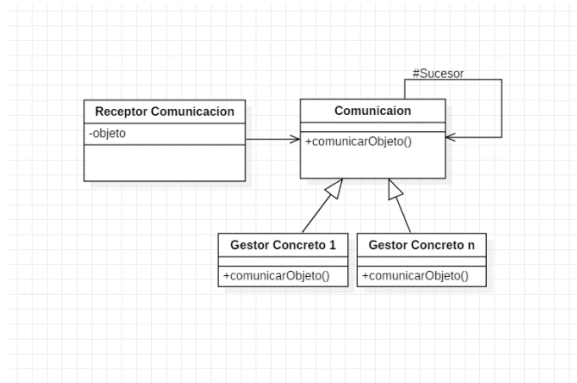
Práctica 1: Captura y Representación de Decisiones de Diseño



Mediante el patrón Facade proveeremos de una interfaz unificada simple para acceder a cada grupo de interfaces del gestor.

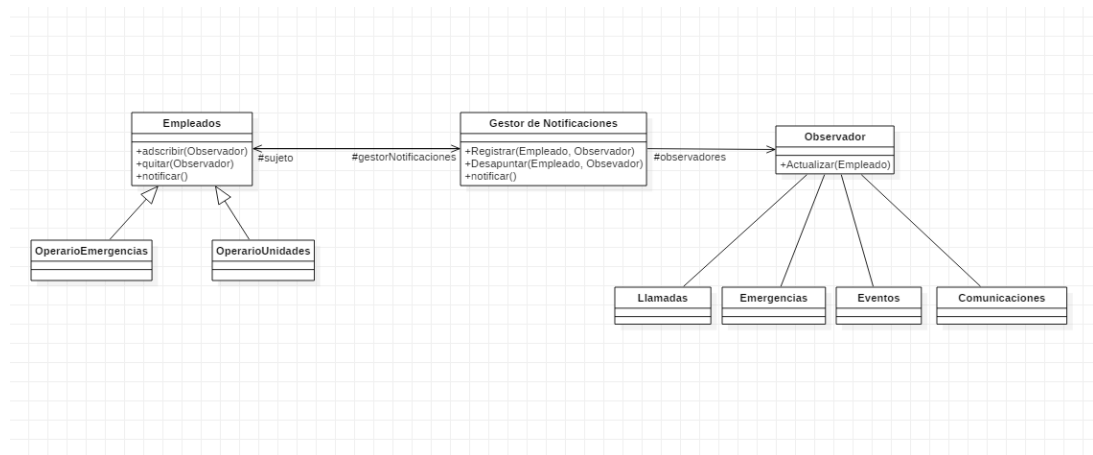
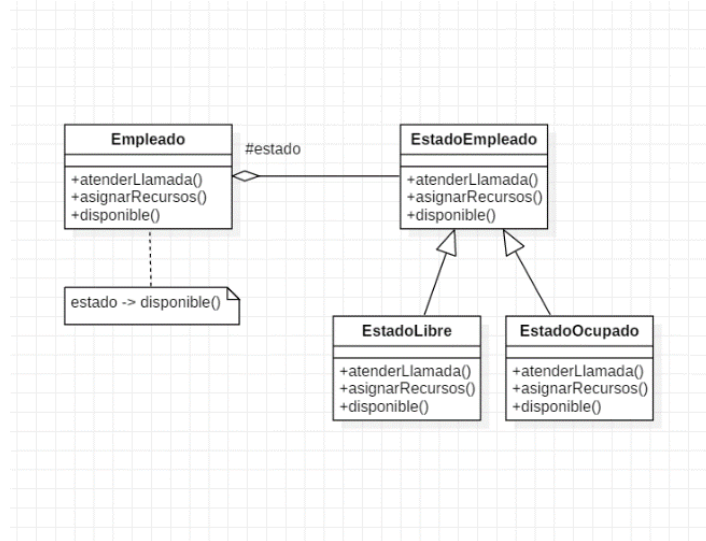


Todo tipo de comunicación entre objetos de diferentes clases deberán llevar el patrón Chain of Responsibility para definir una estructura común en los mismo.



En el caso de los empleados, modificar su comportamiento, en función de si están libre o no, utilizando el patrón state. También usaremos el patrón Observer para notificar los cambios de estos empleados a todos los objetos que lo necesiten saber.

Práctica 1: Captura y Representación de Decisiones de Diseño



Las comunicaciones a los usuarios suscritos se harán mediante el patrón Publish-Subscribe, así, se les notificará cambios no programados. Deberemos regular un sistema de suscripciones para usuarios, que les notificará eventos en tiempo real a Smartphones y tablets.

