StakeInteraction

Administracion de StakeHolders

Integrantes:

Bolpe Jorge (jorbolpe@gmail.com)

Villa Juan Pablo (juampavilla@gmail.com)

Contenido

Introduccion

Objetivo del proyecto

Diseñar y desarrollar una herramienta que permita registrar y fomentar la interacción de los stakeholders con un proyecto y sus componentes. La misma debe facilitar la integración de otras aplicaciones para realizar análisis de redes. Además debe ser adaptable para permitir una posterior expansión e integración de nuevos módulos.

Funcionalidad de la herramienta y características de diseño

La herramienta será una aplicación web (cliente-servidor), que facilitará el acceso a diferentes usuarios a través del explorador para acceder a sus cuentas individuales.

Cada cuenta permitirá diferentes vistas según el usuario que la utilice. Por un lado una vista completa de la red y sus componentes para el administrador y a su vez una vista limitada para stakeholders y/u otro tipo de usuarios. Las vistas se basarán en la teoría en el capítulo 9 del libro “Building the Architecture Documentation”, y se adaptarán a las necesidades del proyecto a lo largo del ciclo de desarrollo.

Diseño arquitectónico de la aplicación web: *StakeInteraction*

Base Arquitectónica (Gwt)

El proyecto utiliza Google Web Toolkit, el cual es un framework creado por Google que permite ocultar la complejidad de varios aspectos de la tecnología AJAX.

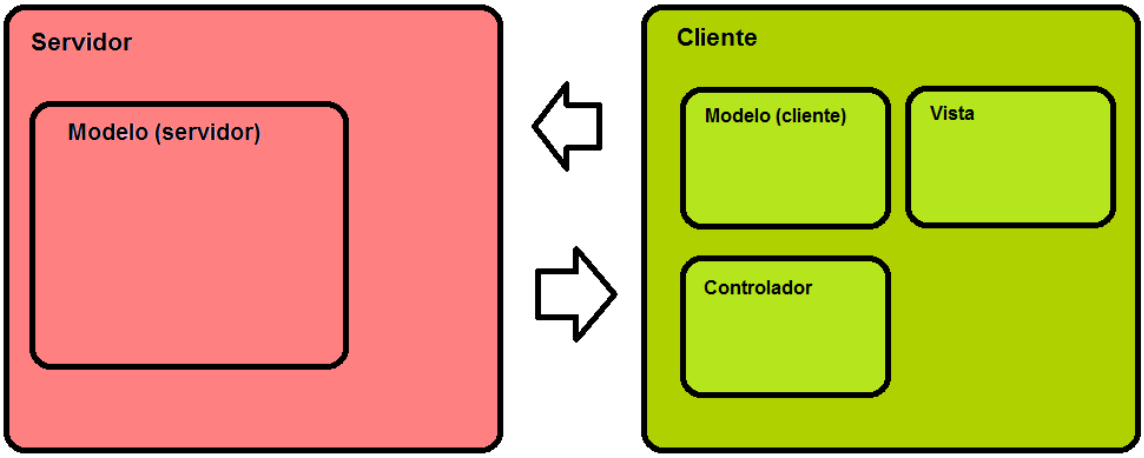
Optamos por esta tecnología ya que permite crear y mantener rápidamente aplicaciones JavaScript con interfaces complejas, pero de gran rendimiento, en el lenguaje de programación Java.

El concepto de Google Web Toolkit es bastante sencillo, básicamente lo que se debe hacer es crear el código en Java usando cualquierIDE de Java y el compilador lo traducirá a HTML y JavaScript.

GWT posee una arquitectura Cliente-Servidor y le permite al servidor ofrecer servicios sincrónicos y asíncronos.

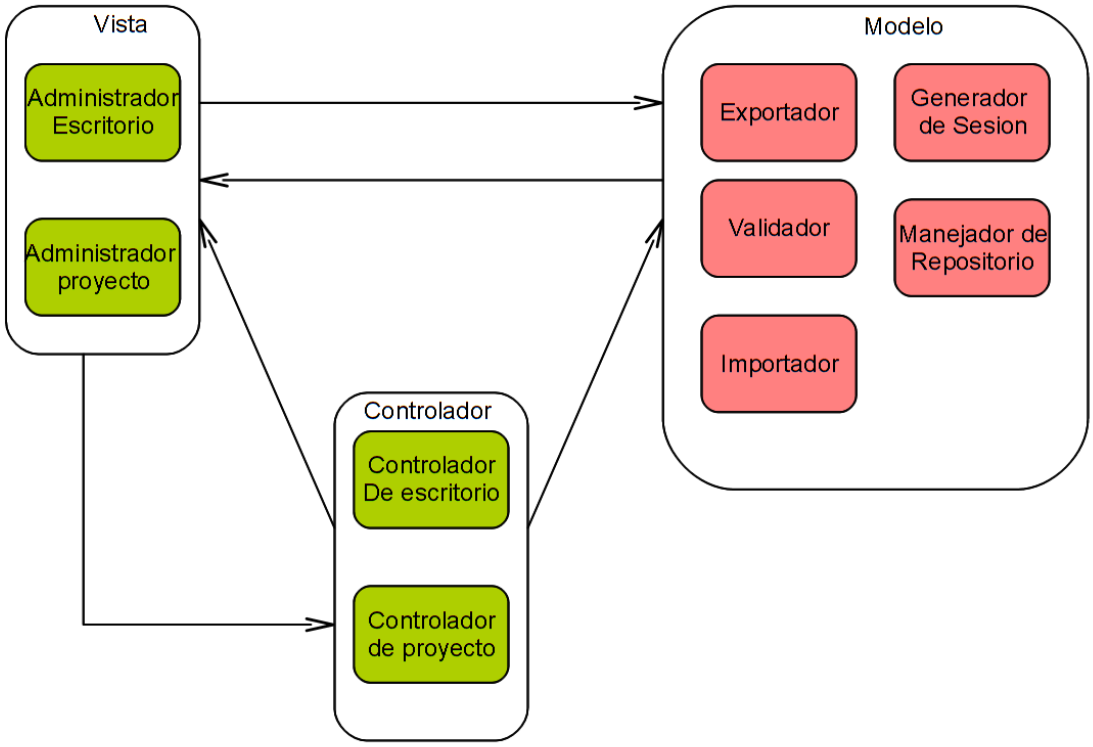
Arquitectura del sistema (MVC)

Nuestro proyecto se monta sobre la arquitectura Cliente-Servidor del GWT, y se presenta en otro nivel nuestra arquitectura Model-View-Controller.



Decidimos instanciar este patrón arquitectónico ya que el mismo se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El "modelo" es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el "controlador" es el responsable de recibir los eventos de entrada desde la vista.

Arquitectura general con componentes



Descripción de los componentes

Vista

Administrador Escritorio: es el encargado de presentar la pantalla de login y cargar el escritorio correspondiente a cada usuario.

Administrador de proyecto: muestra una interfaz que permite modificar el proyecto, teniendo en cuenta los permisos del usuario que se encuentra logueado.

Controlador

Controlador de escritorio: se encarga de capturar los eventos de usuario que se realizan en el escritorio.

Controlador de proyecto: se encarga de capturar los eventos de usuario que se realizan en la pantalla que administra el proyecto.

Modelo

Exportador: Componente encargado de la lógica para exportar el grafo del proyecto para cada usuario particular.

Importador: Componente que contiene toda la lógica para importar grafos del proyecto

Validador: Componente que provee de la lógica para controlar y actualizar los cambios realizados por cada usuario al grafo del proyecto.

Generador de sesión: se encarga de mantener una sesión por cada usuario que se encuentra en el sistema, ofreciéndole los servicios que permiten modificar el proyecto, según el tipo de cuenta que el usuario posea.

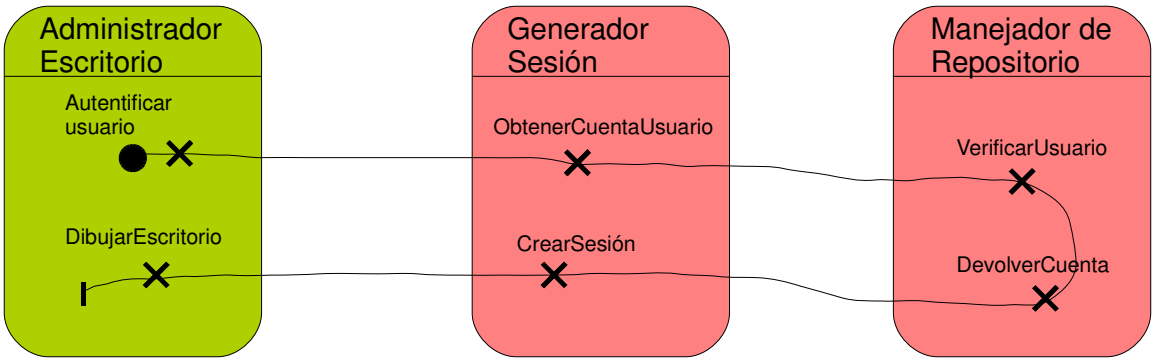
Manejador de repositorio: ofrece los servicios de almacenamiento y obtención de datos de los proyectos y usuarios.

Escenarios

En la siguiente sección plantearemos algunos escenarios relevantes al sistema, con sus correspondientes diagramas de casos de usos, que intentaran explicar el comportamiento de la aplicación, y su interacción entre los componentes arquitectónicos.

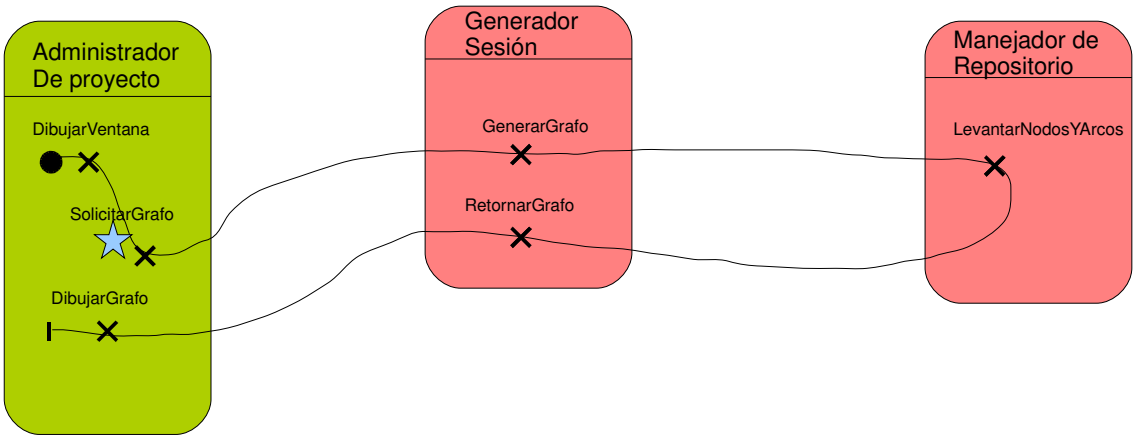
Escenario 1

Un usuario se encuentra en la pantalla de Login, ingresa sus datos, sus datos son autentificados y se le muestra su escritorio correspondiente.



Escenario 2

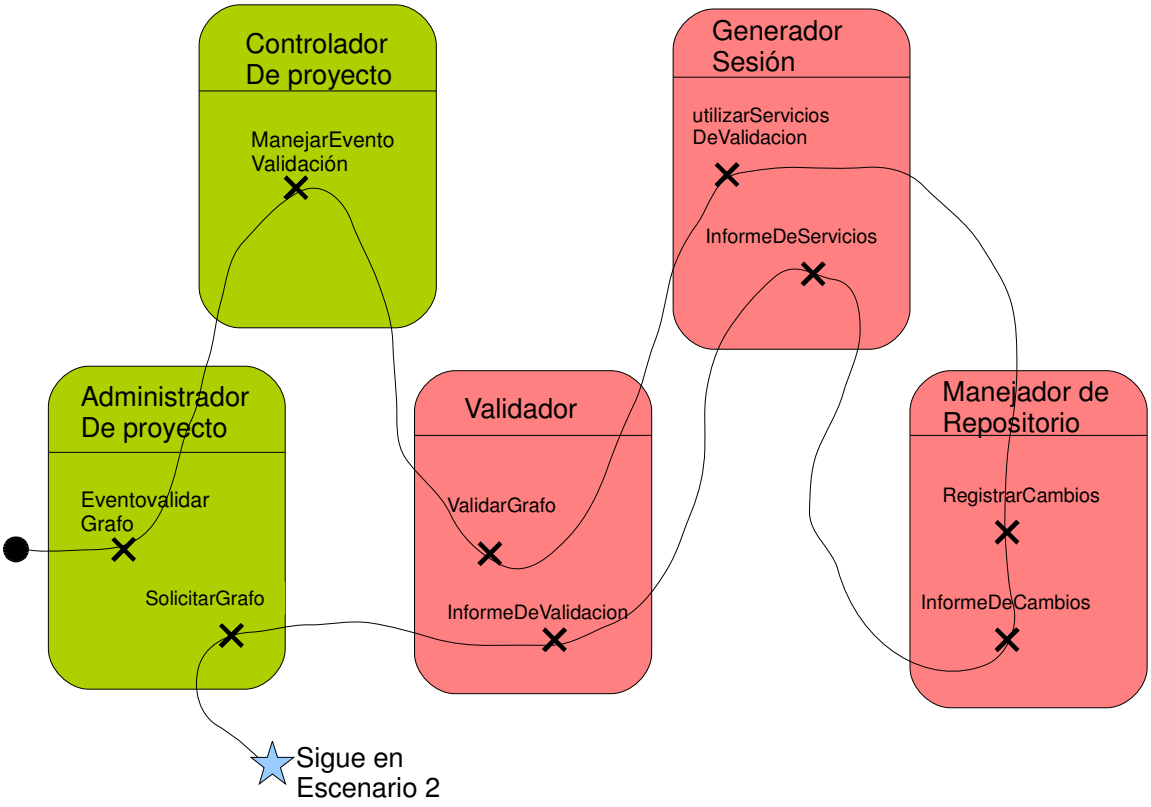
Un usuario luego de abrir el administrador de proyecto, este se carga y se le muestra el grafo correspondiente a su cuenta.



Luego de dibujar la pantalla el programa de forma asincrónica solicita un servicio al servidor, cuando el servidor entrega su respuesta, el administrador dibuja el grafo en pantalla.

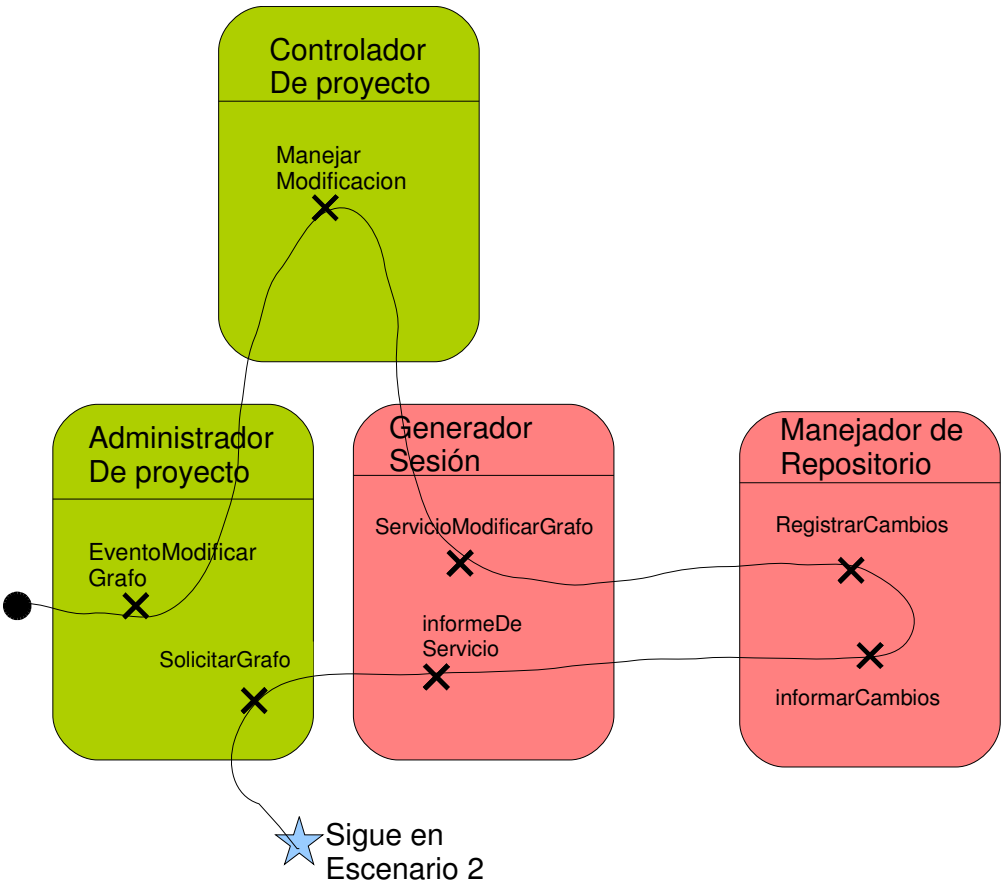
Escenario 3

Un administrador dispara el evento de validar los cambios que han realizados los demás usuarios, entonces se registran los cambios en el grafo.



Escenario 4

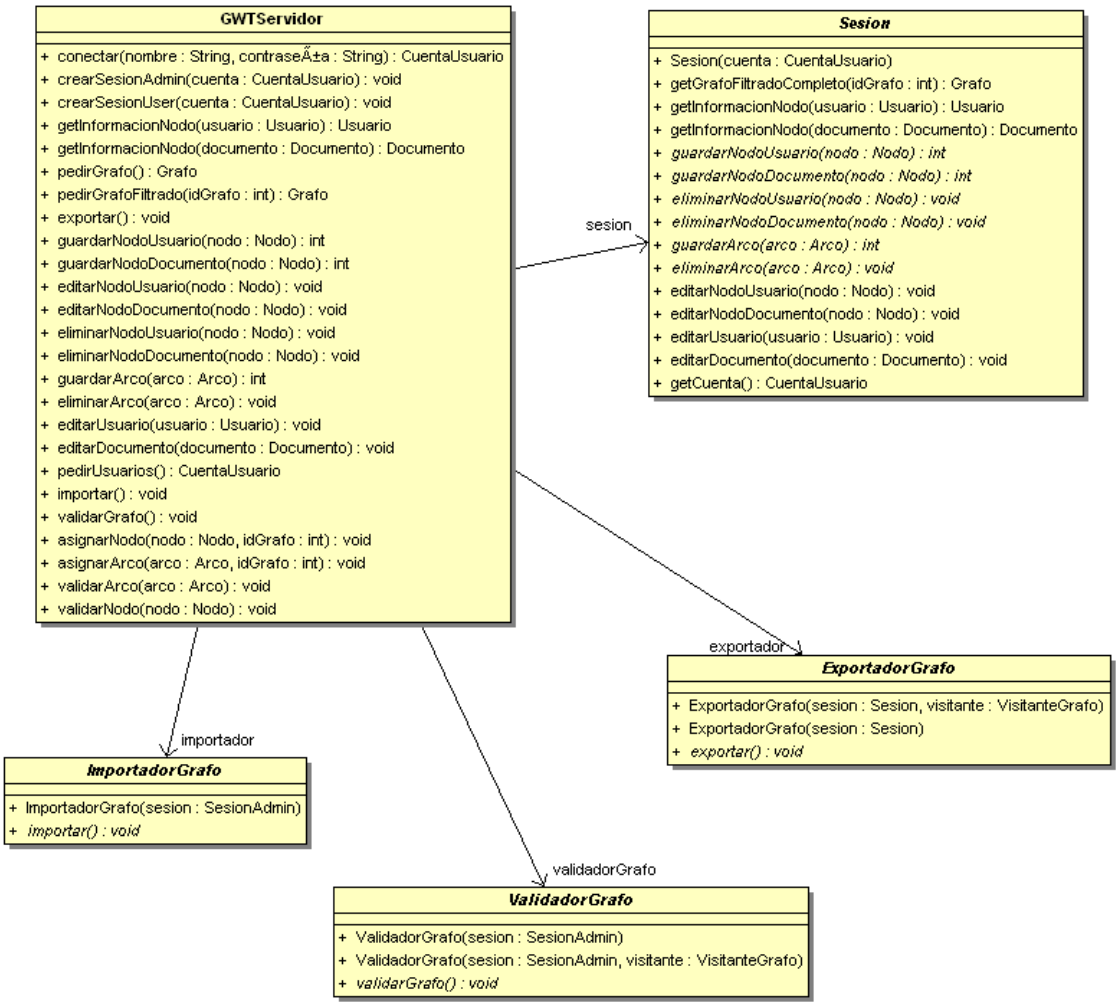
Un usuario/administrador realiza cambios (crear,eliminar,modificar) en los componentes del grafo (documentos , usuarios o conexiones). Luego se redibujar el grafo con los cambios correspondientes.



Diseño orientado a objetos

Modelo(Servidor)

Posee una sesión que provee servicios al modelo cliente y además a otras clases como un importador, exportador y validador de grafo que utilizan dicha sesión.



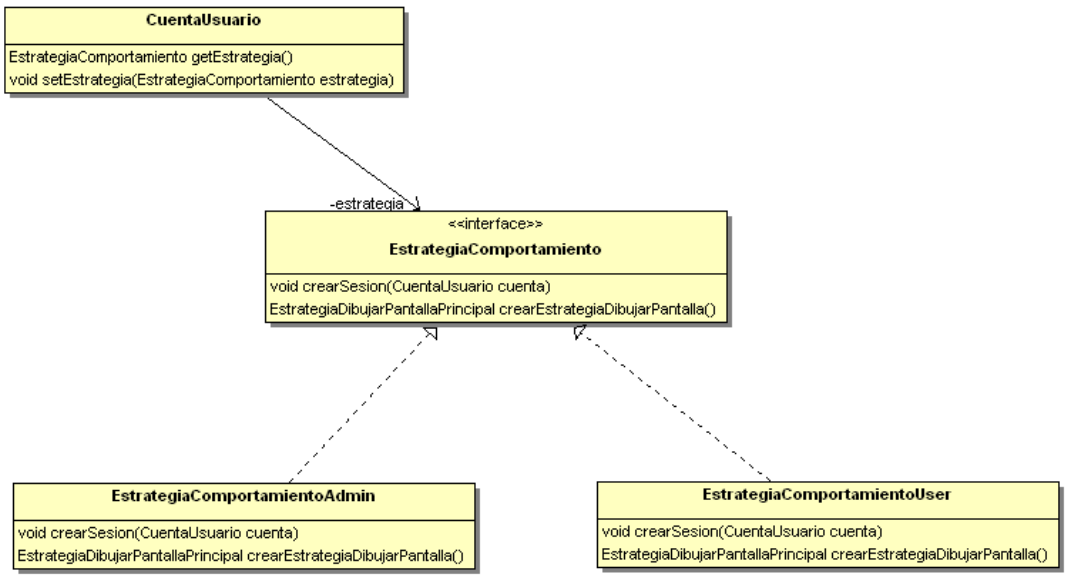
Estrategia comportamiento de la cuenta de usuario

La cuenta de usuario es creada al momento de la conexión. Esta posee una estrategia de comportamiento que es la encargada de crear la sesión y la pantalla inicial que vera el usuario con sus distintas opciones ya sea usuario común o administrador.

La pantalla es la interfaz con la cual se comunica el usuario para utilizar los servicios brindados por la sesión correspondiente.

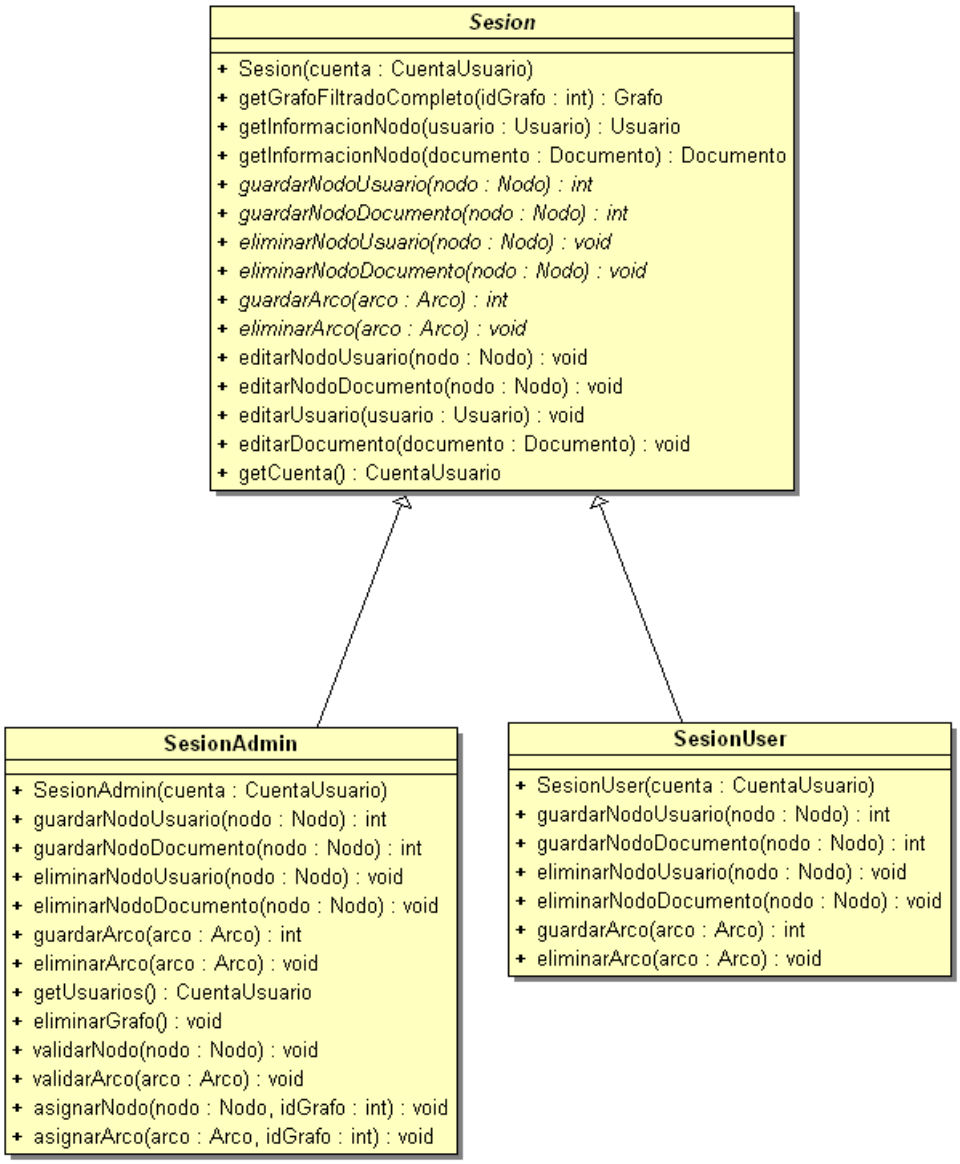
La sesión es la encargada de definir los servicios que un usuario puede solicitar

Dependiendo de si la cuenta de usuario es de administrador o de un usuario común, se creará una nueva sesión correspondiente al tipo de usuario (Sesión, Admin o SesiónUser respectivamente) . A su vez, dependiendo del tipo de usuario se creará una nueva pantalla principal en la cual se cargarán las distintas opciones que tendrá accesibles en pantalla el mismo.



Sesión

Esta clase es la encargada de definir los servicios que un usuario puede solicitar al programa y otros que permiten extender el programa eventualmente.



Define los siguientes métodos comunes, los cuales son compartidos entre un usuario común y un administrador.

Los más importantes son:

- *getGrafoFiltradoCompleto()*
- *Usuario getInformacionNodo(Usuario usuario)*

- *Documento getInformacionNodo(Documento documento)*

El primer método sirve para obtener el grafo del usuario y los otros para obtener la información completa de un nodo ya sea un usuario o un documento.

Por otra parte define métodos abstractos que luego serán implementados por las clases que hereden de esta.

public abstract int guardarNodoUsuario(Nodo nodo);

public abstract int guardarNodoDocumento(Nodo nodo);

public abstract void eliminarNodoUsuario(Nodo nodo);

public abstract void eliminarNodoDocumento(Nodo nodo);

public abstract int guardarArco(Arco arco);

public abstract void eliminarArco(Arco arco);

Las clases SesionUser y SesionAdmin heredan de Sesión y se encargan cada una de definir el comportamiento del usuario común y del comportamiento del administrador respectivamente.

Hicimos esta diferenciación ya que cada usuario del programa tiene acceso a distintas vistas del grafo del proyecto. Definimos que exista el grafo principal y cada usuario (administrador o usuario común) tiene un grafo asignado que apunta al principal (una lista de nodos y una lista de arcos).

SesionUser

El usuario agrega nodos y arcos al grafo con el campo válido en false que posteriormente serán visualizados por el administrador que será el encargado de validarlos o no.

Además este puede eliminar directamente y modificar los nodos y arcos que todavía no estén validados.

Si el nodo o arco es válido solo lo elimina de su propio grafo, es decir elimina solo la referencia al mismo.

SesionAdmin

El administrador agrega nodos y arcos con el campo válido en true directamente del grafo principal.

Por otro lado el administrador elimina directamente del grafo principal, es decir que elimina el objeto y todas las referencias de los usuarios al mismo.

Además la clase SesionAdmin agrega la siguiente funcionalidad:

- Validar nodos y arcos. Estos métodos habilitan validar nodos y arcos manualmente al usuario, y por otro lado pueden ser usados por una clase de visitante concreto del grafo para validar los arcos y los nodos según algún criterio a definir.

- Asignar nodos y arcos a los diferentes grafos de cada usuario. Un nodo o arco inválido no puede ser asignado a un usuario.

- Eliminar el grafo principal. Este método se encarga de eliminar el grafo del proyecto y todas las referencias al mismo.
  Permite eliminar el grafo para posteriormente importar o cargar otro manualmente.

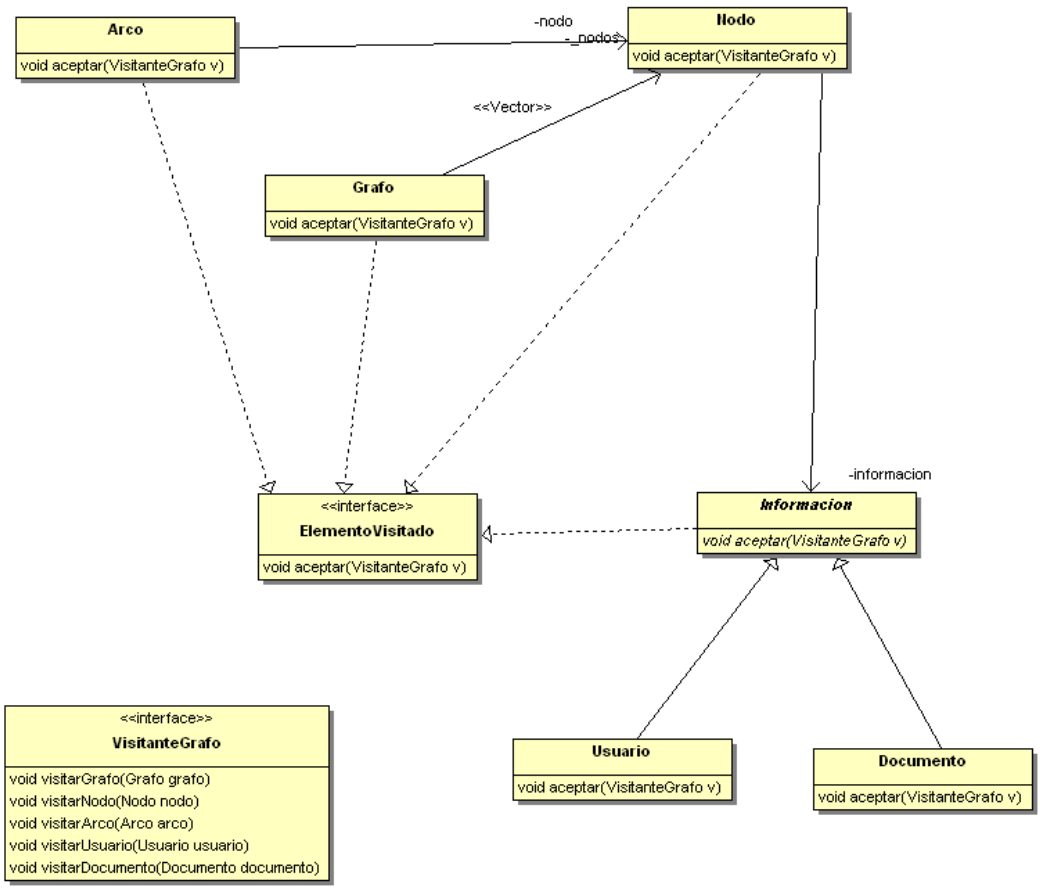
Visitantes Grafo

Utilizamos el patrón Visitor, ya que es una forma de separar un algoritmo de la estructura de un objeto, en nuestro caso el grafo.

Permite agregar operaciones a la estructura del grafo.

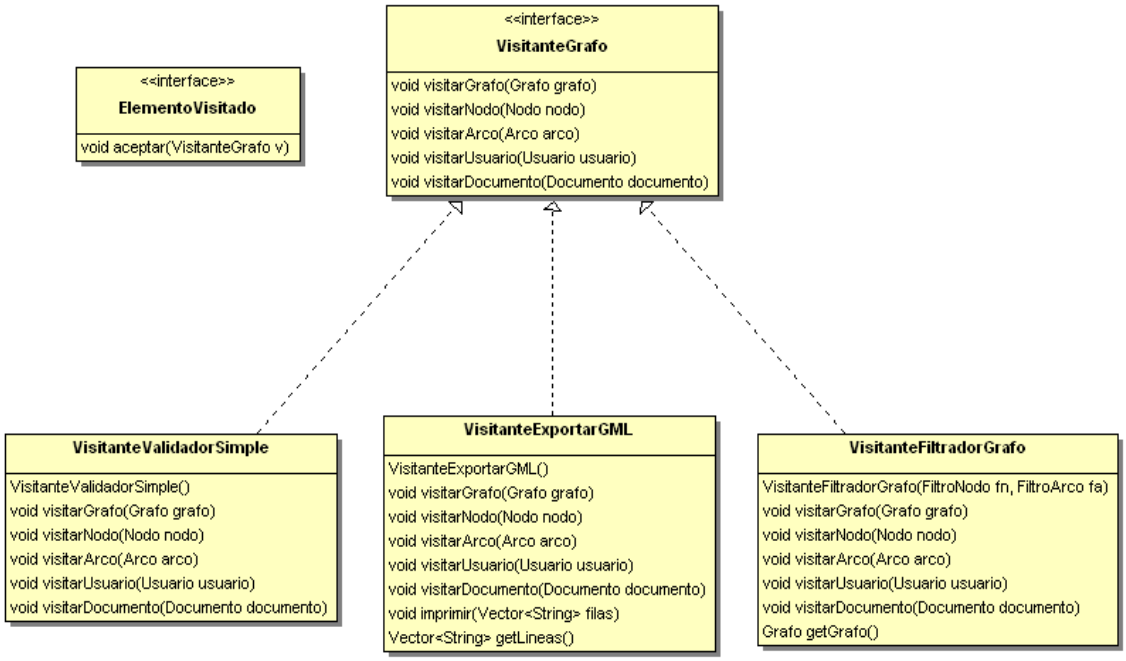
La idea básica es que se tiene un conjunto de clases elemento que conforman la estructura dl grafo.

- Cada una de estas clases elemento tiene un método aceptar (accept()) que recibe al objeto visitante (visitor) como argumento.
- El visitante es una interfaz que tiene un método visit diferente para cada clase elemento.
- Por tanto habrá implementaciones de la interfaz visitor de la forma: visitorClase1, visitorClase2... visitorClaseN.
- El método accept de una clase elemento llama al método visit de su clase.



- Las clases concretas de un visitante pueden entonces ser escritas para hacer una operación en particular.

Definimos 3 clases:



VisitanteValidadorSimple:

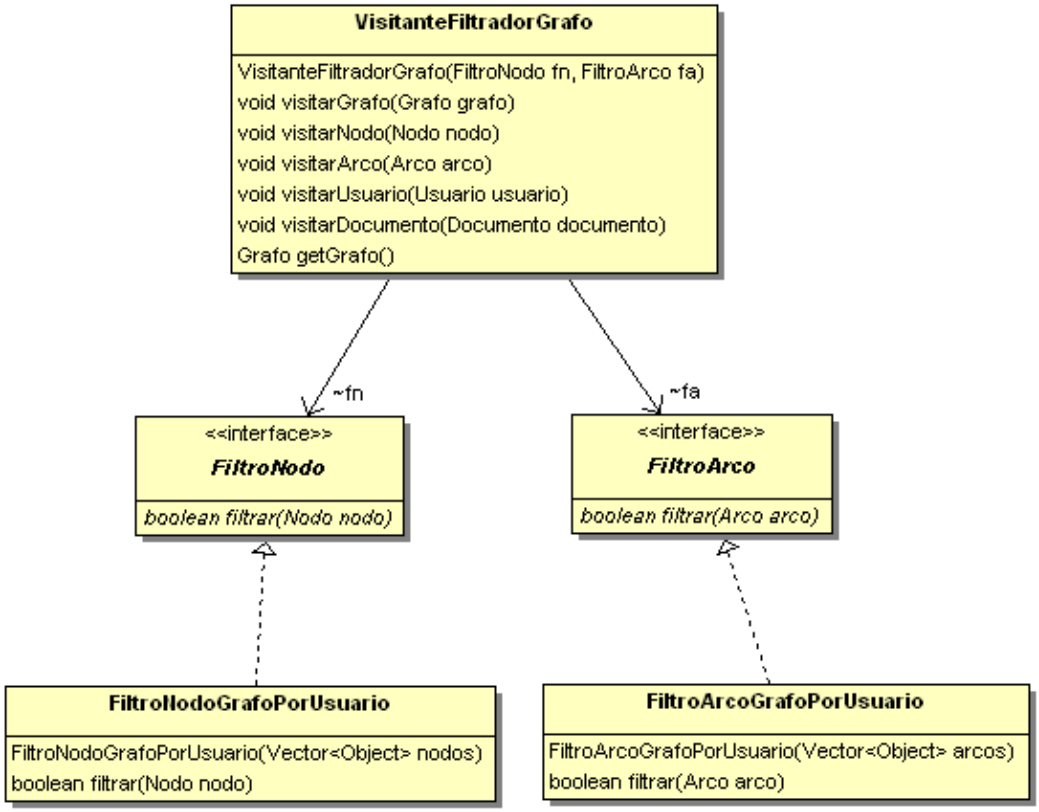
- Se encarga de validar todos los nodos y arcos cuyo estado es inválido. Básicamente es un ejemplo de cómo validar los cambios realizados en el grafo por los usuarios. Se pueden definir distintos validadores con diferentes criterios para validar. Para validar necesita tener acceso a un objeto sesionAdmin que provee los métodos de validar nodo y validar arco.

VisitanteExportarGML:

- Clase encargada de recorrer el grafo y presentar en el formato ".gml" la información que este contiene. Para esto utiliza el método getGrafoFiltradoCompleto provisto en la clase Sesion.

VisitanteFiltradoGrafo:

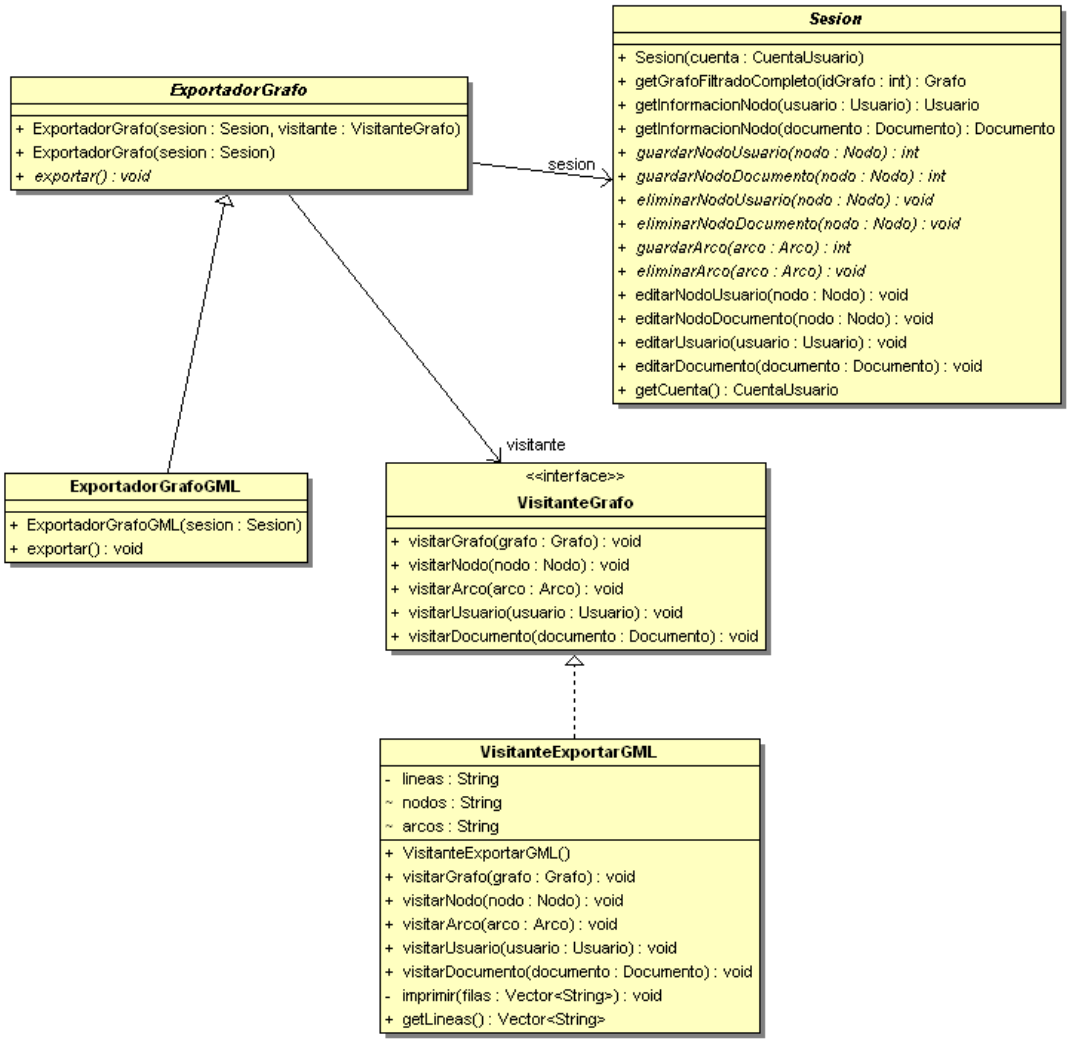
- Utilizando un filtro de arcos y un filtro de nodos, esta clase se encarga de recorrer el grafo entero y solo devolver los nodos y arcos que cumplan dichos filtros.



Exportar Grafo

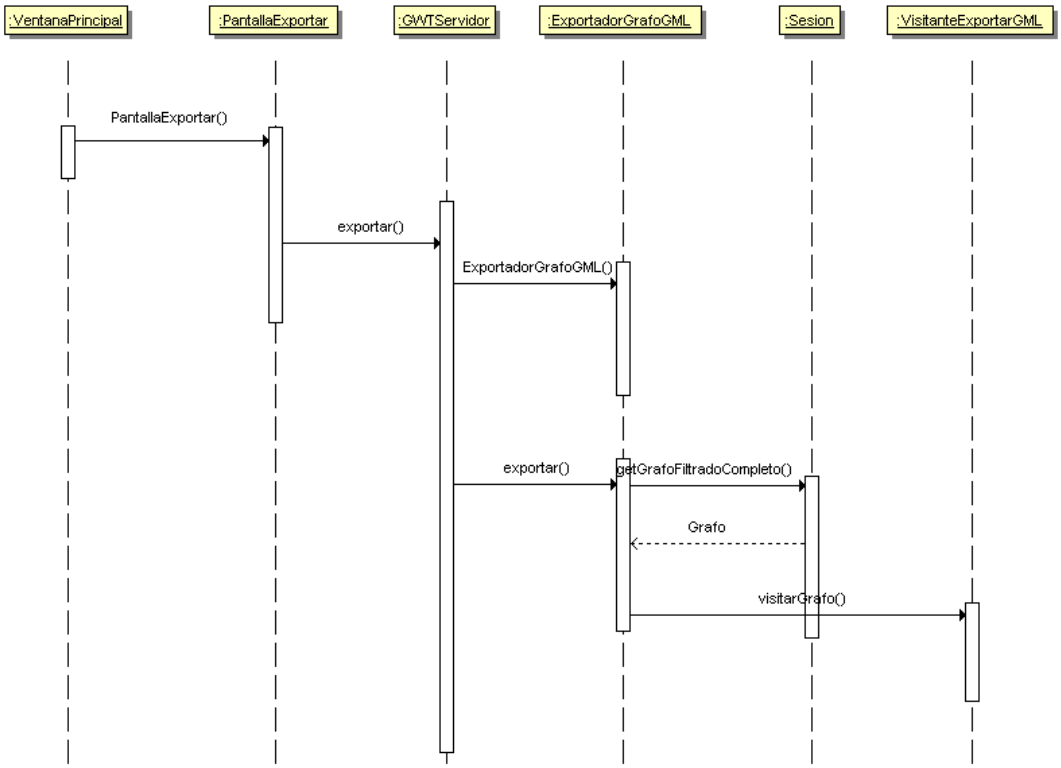
Definimos una clase abstracta para exportar el grafo, la cual utiliza un objeto Sesion para obtener el grafo y un objeto visitanteGrafo para ir recorriendo el mismo. Posee el método abstracto exportar, el cual será redefinido por las clases que hereden de esta y definan el comportamiento de cada exportador concreto.

En nuestro caso definimos un exportador gml, el cual utiliza el visitante **VisitanteExportarGML**.



A continuación modelamos la interaccion entre los distintos objetos:

Diagrama de secuencia (exportar grafo)

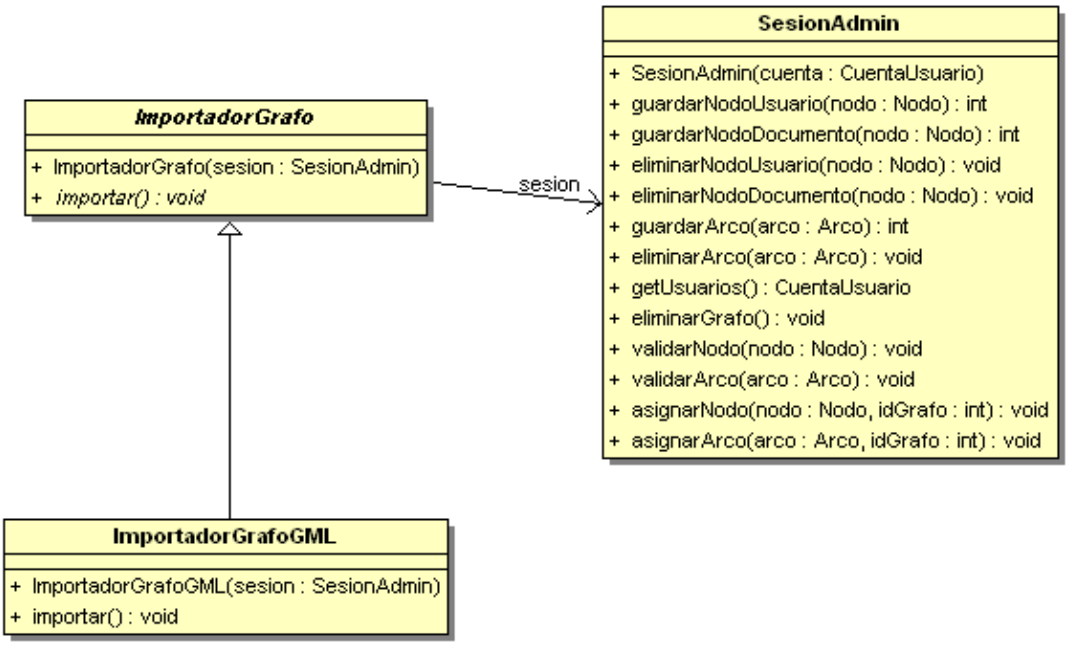


Importar Grafo

Definimos una clase abstracta para importar el grafo que posee el método abstracto importar, el cual será redefinido por las clases que hereden de esta y definan el comportamiento de cada importador concreto.

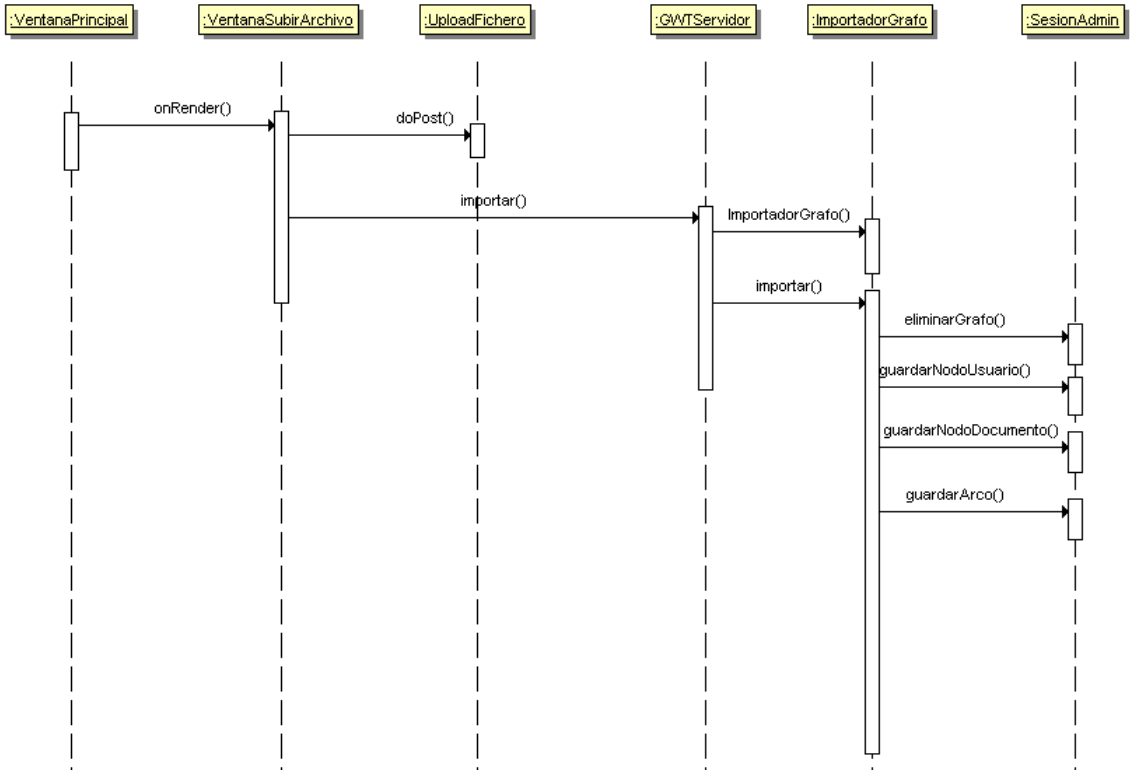
Esta clase utiliza un objeto SesionAdmin que le permite manipular el grafo principal, eliminándolo para luego ir agregando los nodos y arcos con los métodos eliminarGrafo() y guardarNodoUsuario(Nodo), guardarNodoDocumento(Nodo) y guadarArco(Arco) respectivamente.

En nuestro caso definimos un importador gml, el cual lee de un archivo con extensión gml, lo interpreta y lo carga en la base del programa.



A continuación modelamos la interaccion entre los distintos objetos:

Diagrama de secuencia(importar grafo)

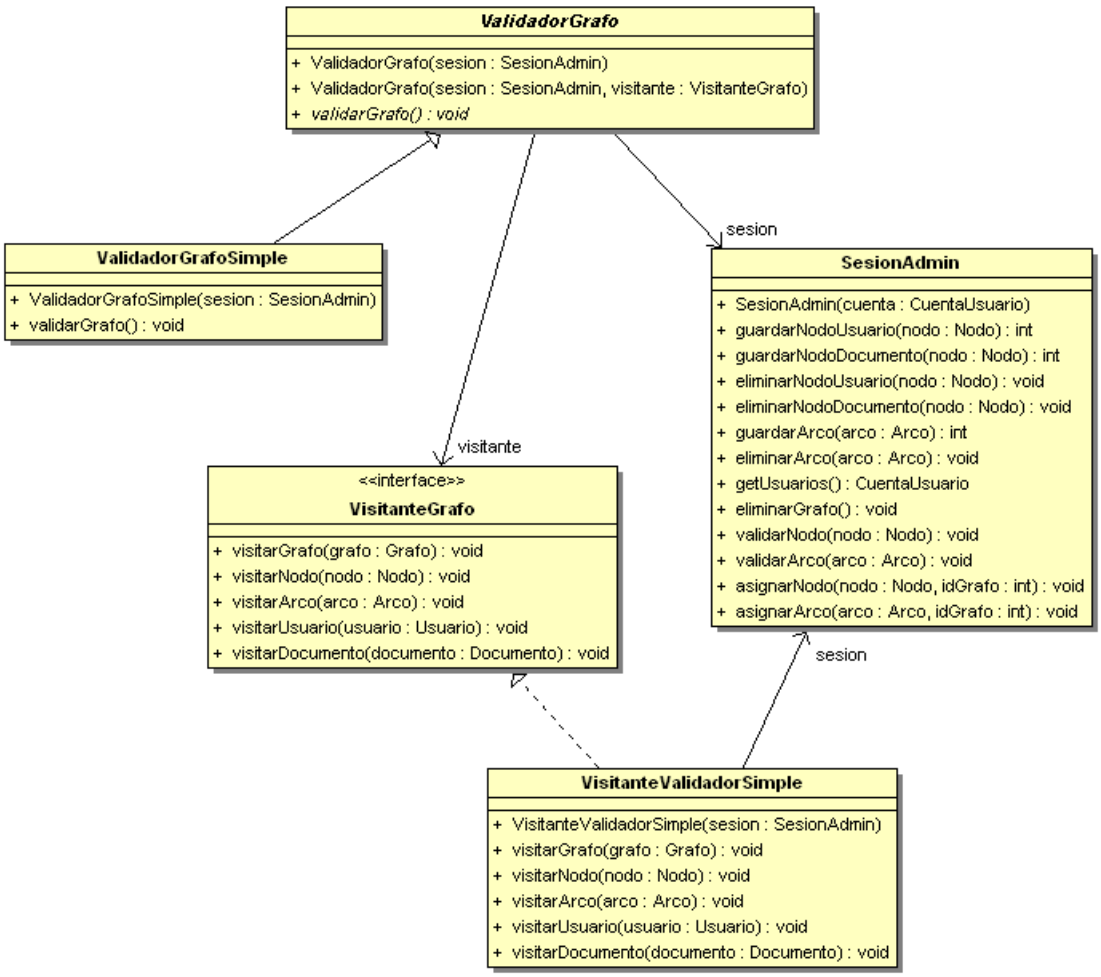


Validar Grafo

Definimos una clase abstracta para validar el grafo, la cual utiliza un objeto Sesionadmin para obtener el grafo y un objeto visitanteGrafo para ir recorriendo el mismo y (utilizando la SesionAdmin) validar los arcos y los nodos inválidos.

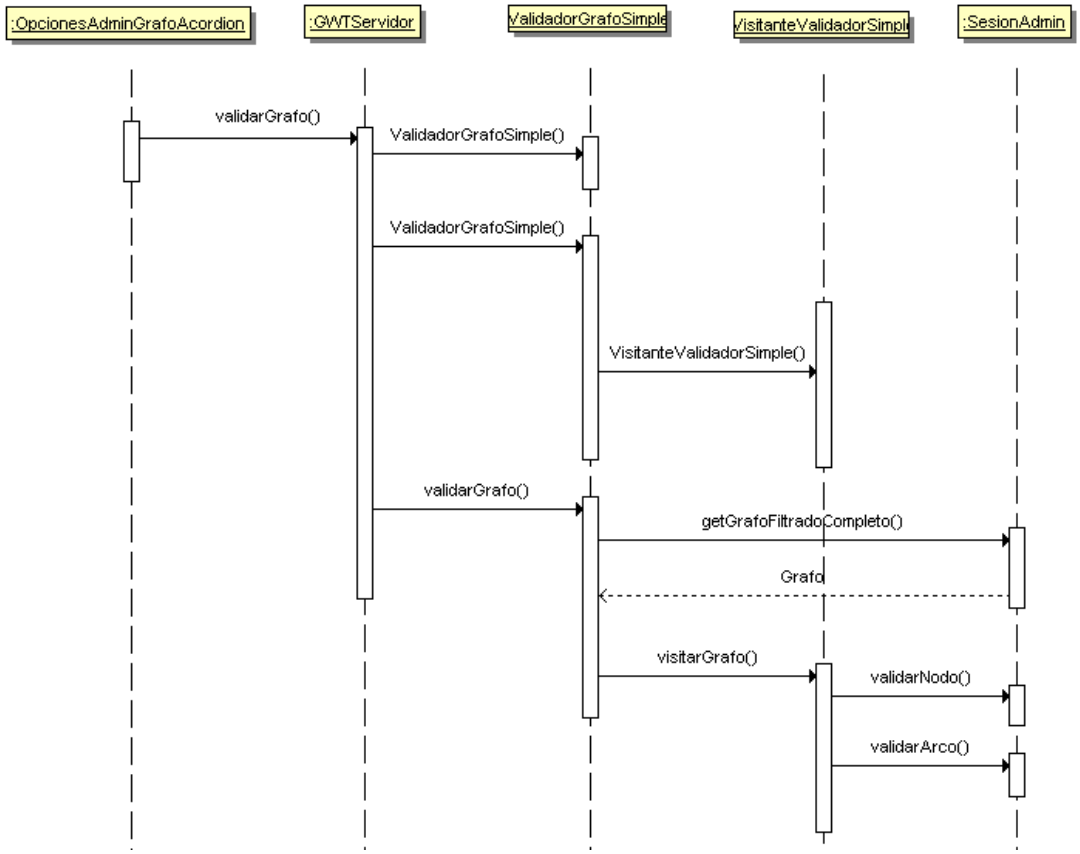
Posee el método abstracto validar en el cual se definirá el comportamiento en una clase concreta de validador.

A modo de ejemplo definimos un validador simple que utiliza el visitante ***VisitanteValidadorSimple.***



A continuación modelamos la interaccion entre los distintos objetos:

Diagrama de secuencia (validar grafo)



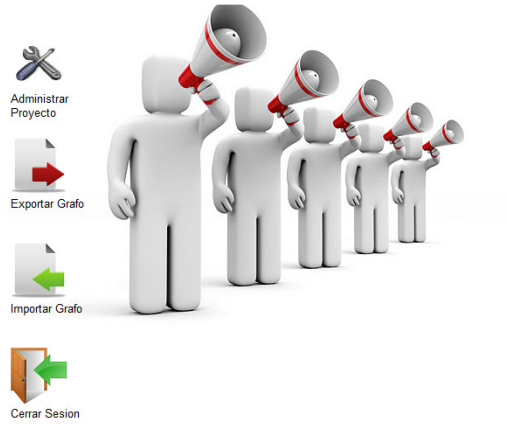
Modelo (Cliente)

Los principales objetos del modelo en la parte del cliente son: una “cuenta” y un “grafo” que obtiene del servidor.

- La cuenta ofrece servicios que se implementan de diferente forma si es una cuenta de Administrador o de usuario.
- El grafo es la estructura de datos que se representa en pantalla

Vista

Esta conformada por una pantalla principal, que ofrece la interfaz gráfica que permite acceder a los servicios que tienen asignados los usuarios o administradores. Presenta una apariencia similar a la del escritorio del Sistema operativo Windows.

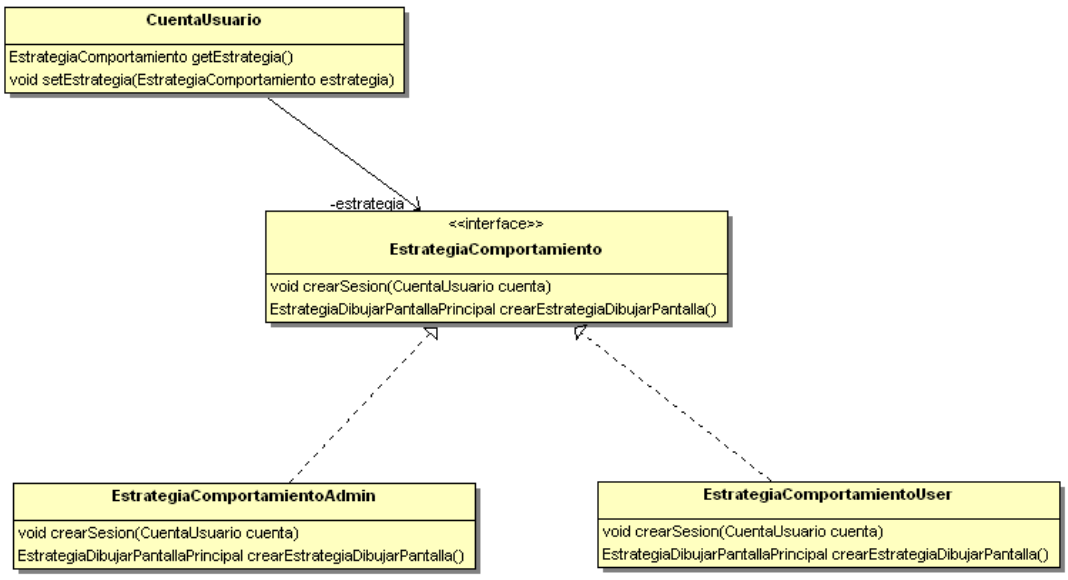


Patrón strategy

Estrategia Comportamiento

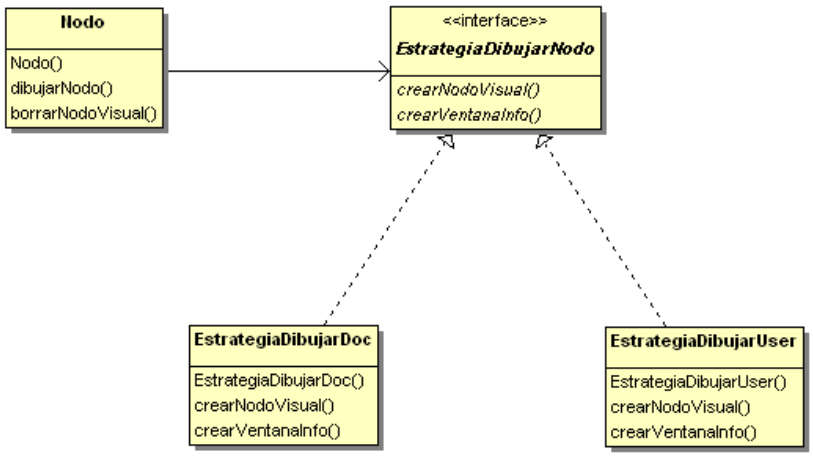
Cada Cuenta de Usuario posee una Estrategia comportamiento que permite mostrar la pantalla correspondiente según sea un administrador o un usuario común.

Al momento de presentar la pantalla, se accede a la cuenta que tiene asignada el usuario, y por medio de su estrategia se presenta el escritorio correspondiente.



Estrategia Dibujar

Cada nodo posee una estrategia que le permite dibujarse en pantalla, ya sea un usuario o un documento. Al momento de dibujar el grafo, se recorre el grafo, y por medio de la estrategia de cada nodo se colocan los dibujos correspondientes en pantalla.

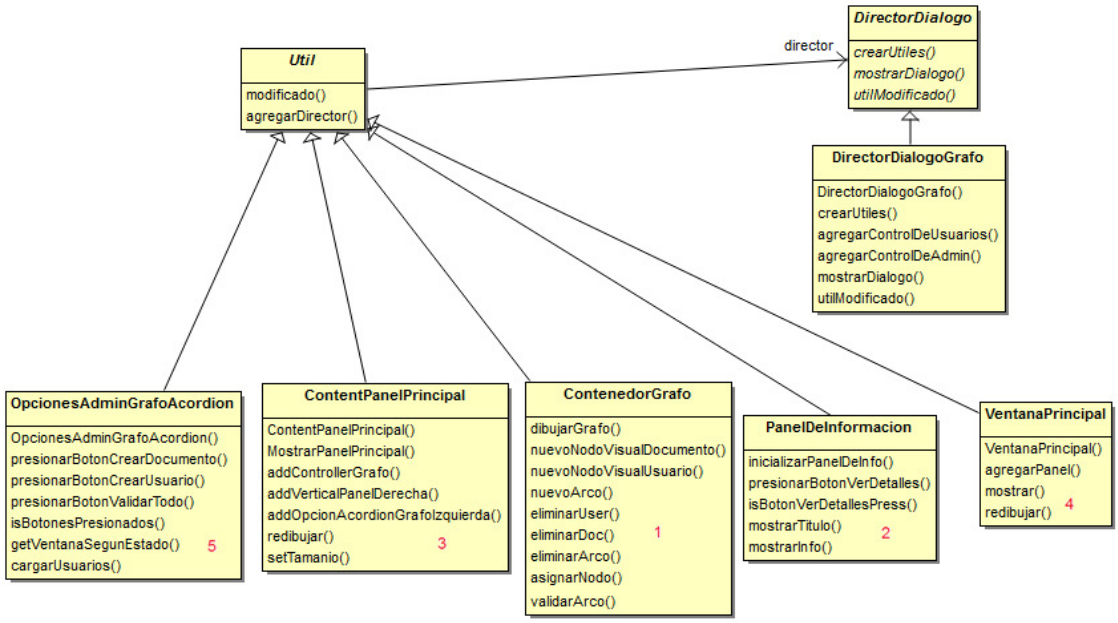


Patrón Mediator

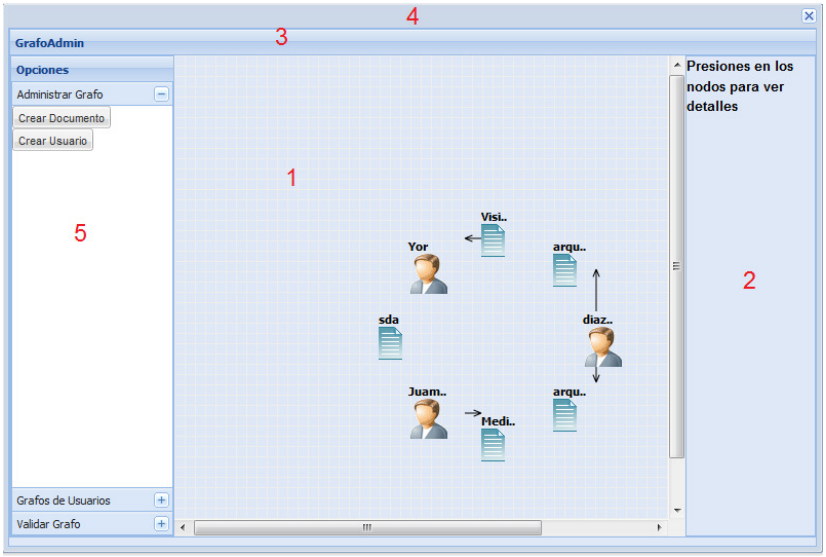
Ventana que permite administrar el proyecto.

La clase DirectorDialogoGrafo se encarga de inicializar todos los componentes de la ventana, además de ubicarlos de manera adecuada.

- Cada componente es un Util, que mediante el método útilModificado(), le avisa al director que debe realizar cambios en la vista.
- Los métodos útilModificado son llamados por los Handlers del sistema, que se encargan de tomar los eventos en pantalla.
- Luego al ocurrir el evento adecuado llaman a UtilModificado, pasándole al director la responsabilidad de invocar los diferentes servicios y de mostrar en pantalla los cambios.



Correspondencia de las clases con los elementos representados en la pantalla(admin proy)



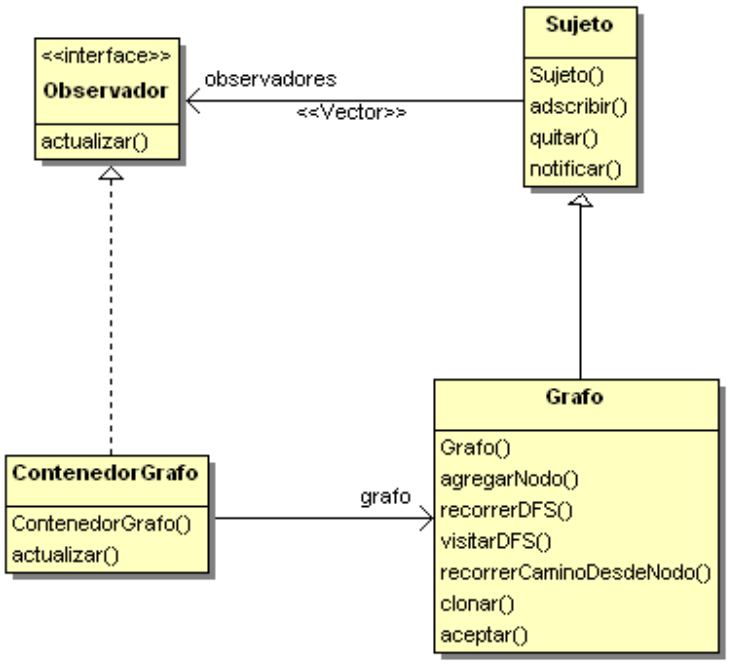
Observer Observable

La clase contenedorGrafo es la encargada de dibujar en pantalla el grafo. Aquí se determinan todos los detalles correspondientes a la ubicación de los widgets en pantalla.

Estrategia simple de dibujo:

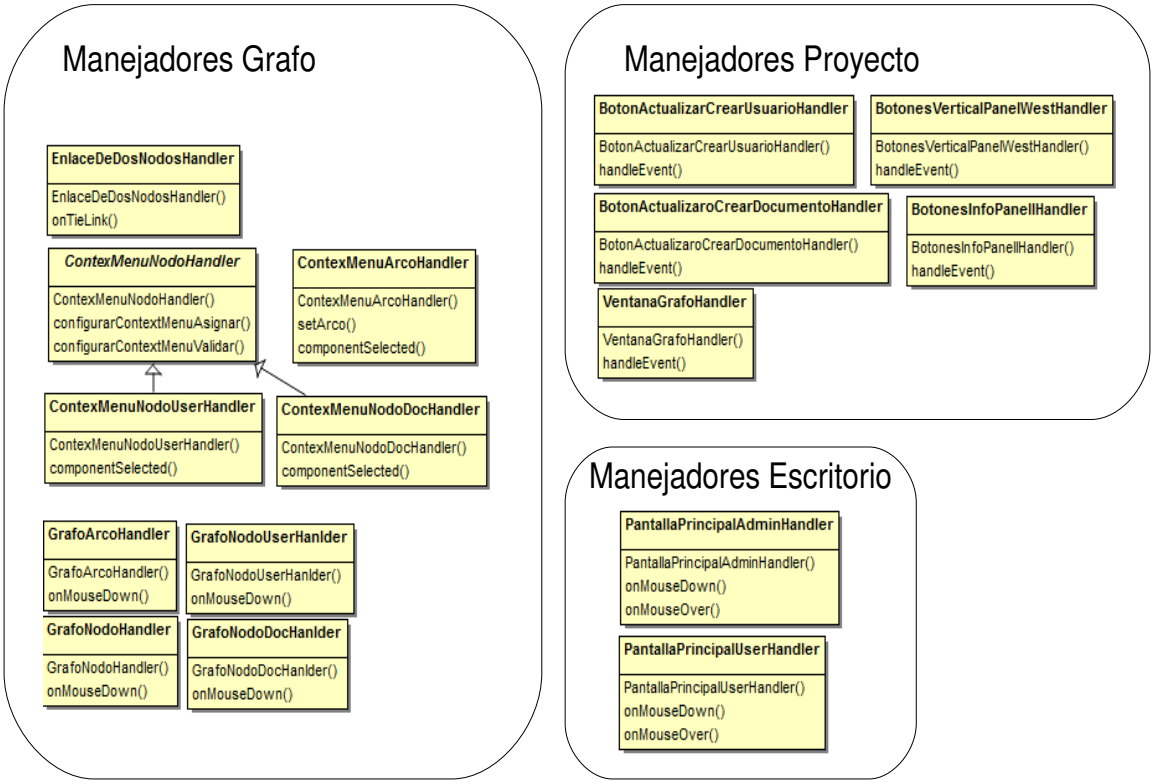
- Se realiza un circulo con todos los nodos.
- Se buscan todas las conexiones.
- Por cada conexión se dibuja un arco.
- Se recorre el grafo marcando los nodos y arcos con la información correspondiente. (nodos o arcos, los cuales pueden ser válidos inválidos o estar asignados a algún usuario.

Cuando ocurre un cambio en el grafo este llama a notificar() , y el contenedor grafo redibuja el grafo en pantalla.



Controlador

Este componente contiene todos los controladores de los componentes del sistema que deben responder a los eventos correspondientes.



Los controladores se encargan de llamar a los servicios correspondientes. La imagen representa que secciones controla cada conjunto de handlers.

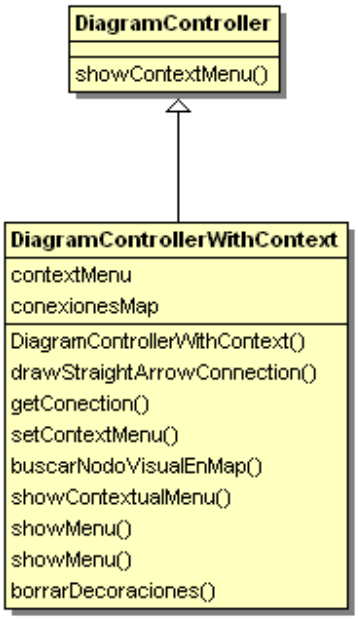
Librerias

Librería Links

http://code.google.com/p/gwt-links/

Permite interconectar widgets en la pantalla por medio de arcos.

Realizamos una extensión de la librería con el fin de controlar los diálogos de contexto de los arcos (al presionar el botón derecho sobre ellos).



Librería GXT

http://www.sencha.com/products/extgwt

Ext. GWT ofrece una extensa librería de widgets de datos de alto rendimiento que son totalmente temáticos y personalizables. Estos incluyen formularios, menús, barras de herramientas, paneles y ventanas entre otros.

ShowCase (Ejemplos)

http://www.sencha.com/examples/#overview

Librería Commons FileUpload

http://commons.apache.org/fileupload/

El paquete Commons FileUpload hace que sea fácil agregar la capacidad de subir archivos a servlets y aplicaciones web.

Conector de Java-Mysql

http://dev.mysql.com/downloads/connector/j/3.0.html

MySQL Connector / J es el driver oficial de JDBC para MySQL.

Apendice

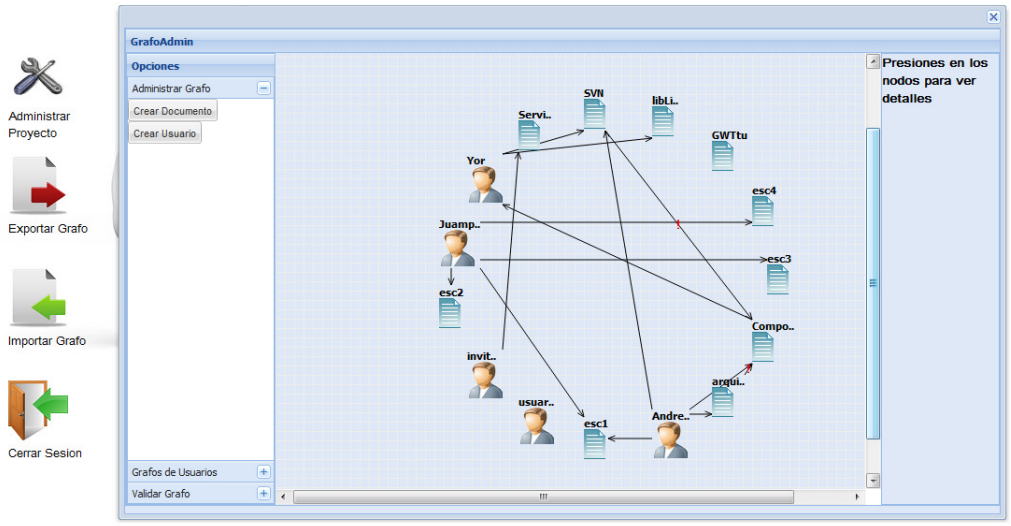
Instanciacion del ejemplo de la red

Para demostrar el funcionamiento de la aplicación cargamos el desarrollo de este mismo proyecto a modo de ejemplo para demostrar la interacción de los stakeholders con un proyecto y sus componentes.

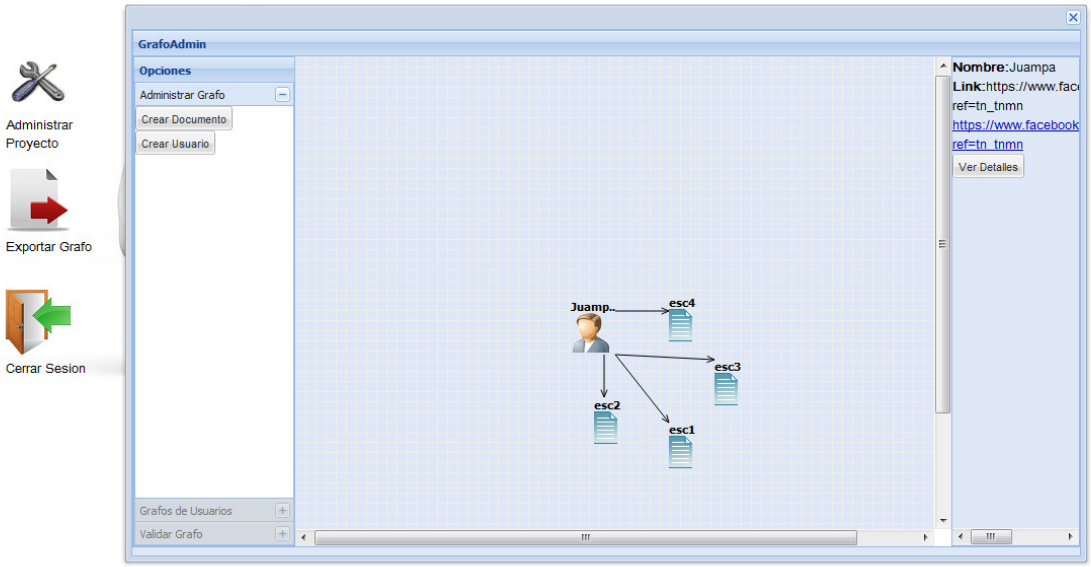
En este caso puntual los stakeholders son: Andres Diaz Pace, Juan Pablo Villa y Jorge Bolpe y como documentos cargamos toda la información relacionada a la documentación de la aplicación.

Se puede acceder a la aplicación con seis cuentas de usuario(andres,juampa,yor,user5,user6) y un administrador(admin1). La contraseña para todos es 4321.

administrador



usuario


Administrar Proyecto
Exportar Grafo
Cerrar Sesion
GrafoAdmin
Opciones
Administrar Grafo
Crear Documento
Crear Usuario
Grafos de Usuarios
Validar Grafo
Juamp..
esc4
esc3
esc2
esc1
Nombre:Juampa
Link:https://www.fac
ref=tn_tnmn
https://www.facebook
ref=tn_tnmn
Ver Detalles

Tutorial de deployment de la aplicación web StakeInteraction

Deployment de la Base de Datos sobre la que corre la aplicación web

Crear tablas e información inicial.

- Correr los siguientes scripts que se encuentran junto con la documentación
- En la documentación se incluye un script llamado *createDB.sql* que crea todas las tablas del programa.
- Se incluye también un script llamado *createUsers.sql* que inserta 3 usuarios, un administrador y dos usuarios regulares.

Conexión con la Base

El programa corre sobre una Base de Datos MYSQL 5.5.14

El programa se conecta con la siguiente información:

- esquema “disenio_schema”
- User:"root"
- Password:"pass"

Estos datos se pueden cambiar desde el código de fuente, en la clase “ManejadorBase” que se encuentra en el paquete:

“org.stakeInteraction.server.Base”

Estructura de la tabla cuenta de usuario

Para poder crear un usuario nuevo hay que tener en consideración la siguiente estructura.

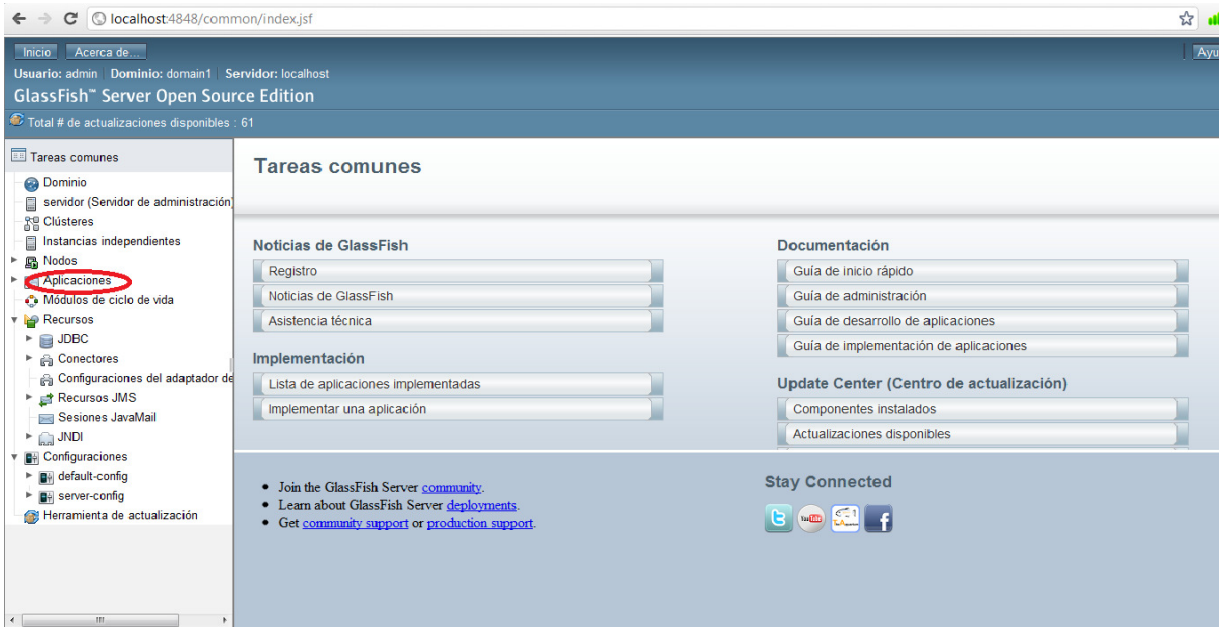
idCuentaUsuario	nombre	contrseña	idGrafo	Tipo
1	admin1	4321	1	admin
2	user1	4321	2	user
3	user2	4321	3	user

IdGrafo: indica el grafo que el usuario tiene asignado. Muchos usuarios pueden tener asignado el mismo grafo.(esta opción podría soportar asignar los grafos a distintos grupos de usuarios).

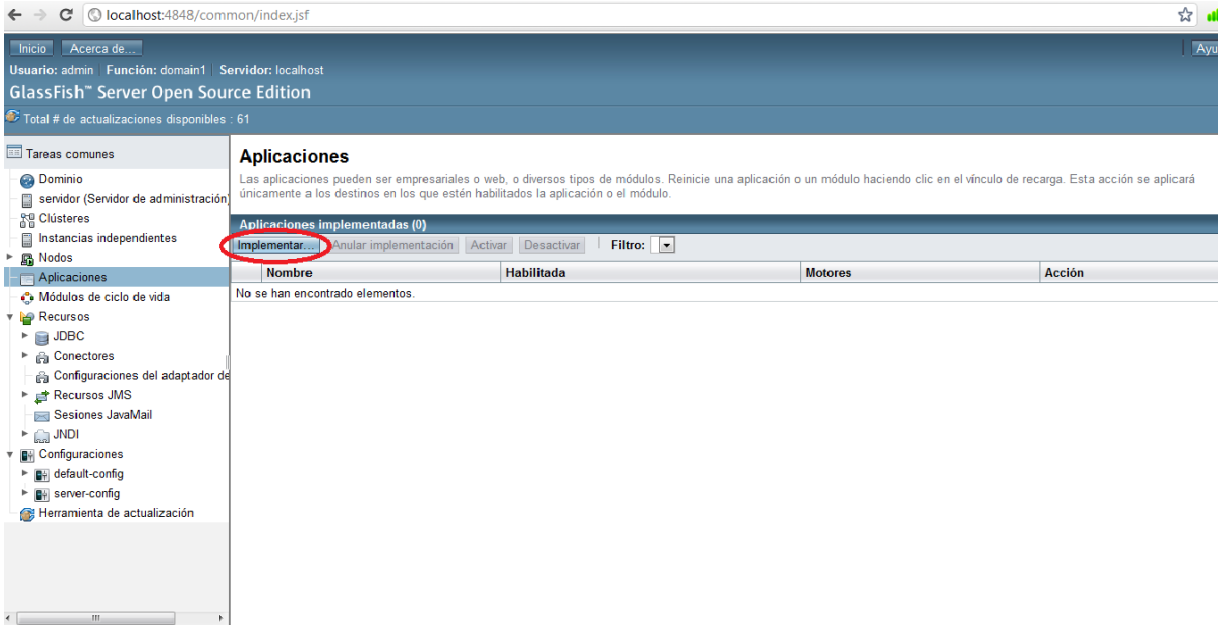
Tipo: puede ser “admin “ o “user”

Deployment de StakeInteraction en un servidor GlassFish

1. Descargar el servidor Glassfish desde el siguiente link. (versión 3.1.2) http://www.oracle.com/technetwork/middleware/glassfish/downloads/ogs-3-1-1-downloads-439803.html
2. Iniciar la instalacion típica asegurándose que los puertos 8080 y 4848 esten libres.
3. Ingresar desde el explorador a la consola del servidor (http://localhost:4848)
4. En el menú de la izquierda, elegir la opción “Aplicaciones”

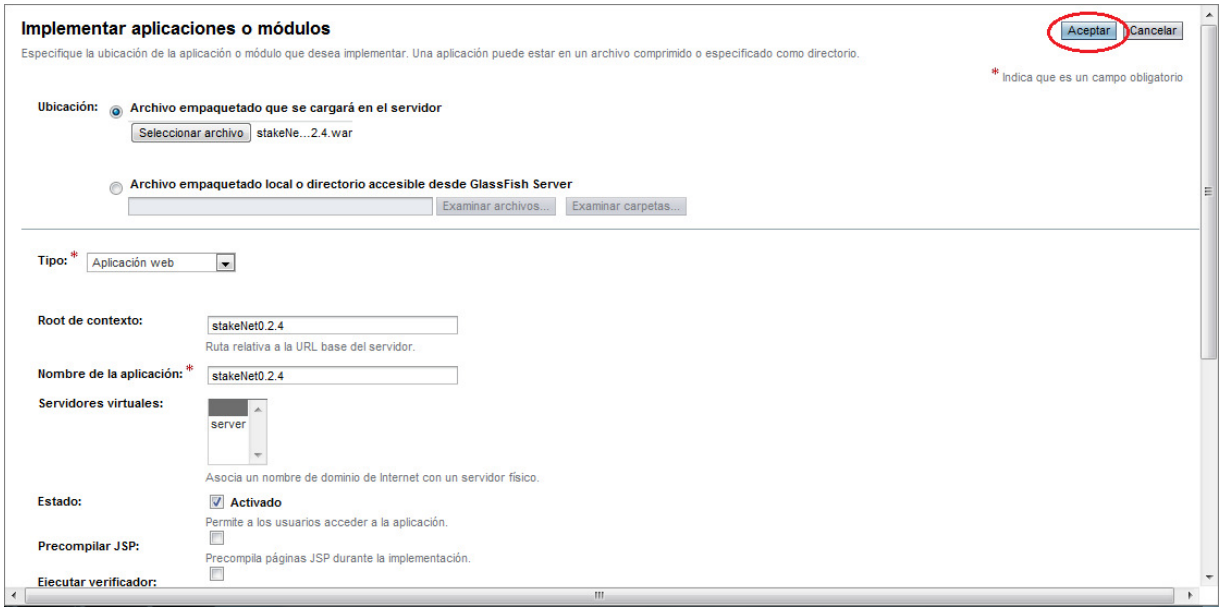


5. Hacer click en el botón implementar

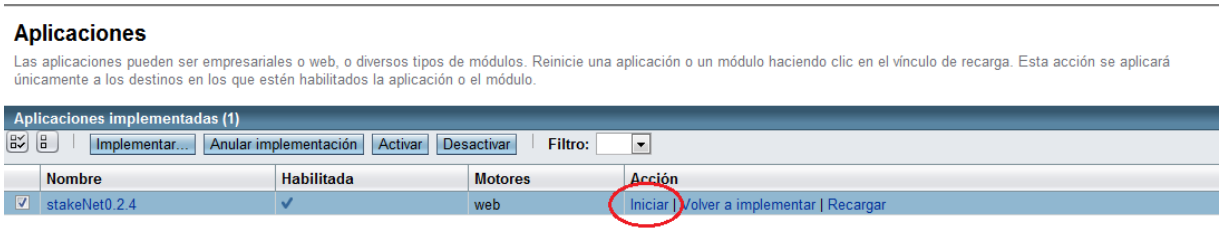


6. Especificar la ubicación de la aplicación o módulo que desea implementar. Hacer click en la opción "**Archivo empaquetado que se cargará en el servidor** " y seleccionar el archivo WAR de la aplicación, que se incluye en la documentación.

7. Aparecerá la siguiente pantalla. Hacer click en el botón “aceptar” arriba a la derecha.



8. Seleccionar la aplicación y hacer click en la opción “iniciar” y se redirigirá automáticamente a la pagina principal de la aplicación.



Otra forma de iniciar el servidor manualmente desde consola

1. Dirigirse al directorio de instalación del glassfish y entrar a la carpeta bin. (\glassfish-3.1.1\glassfish\bin)
2. Ejecutar el archivo “asadmin.bat”, se abrirá la consola y con el comando “start-domain” se inicializa el servidor y con el comando “stop-domain” se finaliza.
3. Luego entrar a la consola de administración (como en el paso 3 de las instrucciones anteriores y seguir los siguientes pasos).

DVD Adjunto

En el DVD le adjuntaremos la siguiente información:

- Código de fuente.
- El ejecutable WAR.
- Script de creación de la Base de datos.

Además el proyecto se encuentra en:

http://code.google.com/p/stakeinteraction/