Funcionalidades desarrolladas para extCatastro

Las funcionalidades implementadas para la extensión extCatastro, contienen en términos generales los siguientes desarrollos:

- Selección del área de trabajo
- Herramientas de digitalización
- Implementación de reglas de negocio y topológicas

Para un usuario, el punto de entrada en la extensión son las distintas acciones de los menús. Para el desarrollador, cada una de ellas puede ser vista como una extensión, que se encontrará en el paquete java llamado `es.icarto.gvsig.catastro`.

Como anteriormente se apuntaba, en este paquete se encuentran:

- Extensiones de selección:
  - Selección de la región: clase `SelectionRegionExtension.java`
  - Selección de la manzana: clase `SelectionManzanaExtension.java`
  - Selección del predio: clase `SelectionPredioExtension.java`
  - Resetear la sección: clase `SelectionResetConstantsExtension.java`
- Extensiones de digitalización:
  - Dividir una construcción: clase `ConstruccionDivideExtension.java`
  - Fusionar una construcción: clase `ConstruccionFusionExtension.java`
  - Crear una nueva construcción: clase `ConstruccionNewExtension.java`
  - Crear una nueva manzana: clase `ManzanaNewExtension.java`
  - Dividir un predio: `PredioDivideExtension.java`
  - Fusionar un predio: `PredioFusionExtension.java`
- Extensiones de las reglas:
  - Lanzar las comprobaciones y acciones topológicas y de negocio: clase `ActionDispatcherExtension.java`

A lo largo de las siguientes secciones se explicará en detalle cada una de las anteriores funcionalidades.

Selección del área de trabajo y constantes

Resumen

- Las extensiones de selección, inicializan la herramienta de selección por punto de gvSIG.
- Un listener sobre las acciones de la anterior herramienta se encarga de inicializar la información necesaria sobre el área seleccionada.
- Las estructuras `ConstantManager.java` y `Constants.java` mantienen y permiten consultar la información en cualquier punto del programa.
- El usuario puede ver la información de constantes seleccionadas en la statusbar de gvSIG.

Paquetes java relacionados:

- `es.icarto.gvsig.catastro`: extensiones de selección.
- `es.icarto.gvsig.catastro.constants`: estructuras para mantener la información y rellenarla.
- `config.xml`: para incluir la statusbar de las constantes.

Flujo de la selección

Las extensiones de selección, permiten al usuario seleccionar el área sobre la que van a trabajar, así como mantener esa información a lo largo de la aplicación, para que otras extensiones puedan consultarla (lo que llamaremos constantes de operación).

Para realizar lo anterior, cada extensión usará la información que provee la herramienta de selección por punto de gvsig.

En código, esto se convierte en un listener sobre las acciones de esa herramienta. Así, para tomar la información, cada una de las extensiones mantiene el listener ConstantsSelectionListener, que se encargará de inicializar las estructuras `Constants.java` y `ConstantManager.java`.

Cada una de las extensiones, en su método execute, tendrá pues un código como el siguiente:

```
View view = (View) PluginServices.getMDIManager().getActiveWindow();
MapControl mc = view.getMapControl();
ConstantsSelectionListener csl = new ConstantsSelectionListener(mc);
mc.addMapTool("constantsSelectionManzana", new PointBehavior(csl));
mc.setTool("constantsSelectionManzana");
```

Desde este momento, el usuario tiene a su disposición la herramienta de selección por punto de gvSIG. Una vez pinche sobre una entidad, el listener que se ha inicializado se encargará de setear las constantes necesarias.

El listener de selección

La clase `ConstantsSelectionListener.java` hereda de `PointSelectionListener.java` e implementa su método callback, que será llamado una vez el usuario seleccione una entidad.

El código del núcleo de esta clase es:

```
@Override
public void point(PointEvent event) throws BehaviorException {
    switch (layer) {
        case LAYER_IS_PREDIO:
             // set constants for predio
             break;
        case LAYER_IS_MANZANA:
             //set constants for manzana
             break;
        case LAYER_IS_REGION:
             // set constants for region
             break;
}
```

La herramienta de selección por punto de gvSIG depende de la capa activa. Es por ello que las comprobaciones sobre qué constantes deben inicializarse se realizan mediante la comprobación de la capa activa.

Las constantes

Las constantes de selección se mantienen mediante la estructura `Constants.java`. En esta clase se mantienen información sobre la región, manzana y/o predio seleccionados.

Esta estructura también proporciona acceso a otras variables que pueden ser consideradas constantes pero en este momento son fijas para toda la operación del programa: país, estado, municipio y límite municipal. Estas últimas son ahora definidas como variables estáticas en la clase *Preferences.java*`, pero se han incluido en la estructura de constantes por coherencia. El desarrollador que use esta estructura necesita la información conjunta sin preocuparse del lugar donde ésta sea almacenada.

```
public String getPais() {
   //TODO: este campo se debe recuperar de una tabla de la BD
   return Preferences.PAIS;
}
```

A lo largo de la ejecución del programa, cuando sea necesario leer el valor de

las constantes, se utilizará la estructura `ConstantManager.java`. Esta clase mantiene una variable de clase llamada `constants` (que es compartida por todos las instancias de la clase).
private static Constants constants = null;

Así, aunque `ConstantManager.java` no es un singleton en sí mismo, se comporta como tal, pues devuelve la misma variable constants a cualquiera de sus instancias o la crea si no existe:

```
public Constants getConstants(){
   if(constants == null){
       constants = new Constants();
           return constants;
           }
           return constants;
}
```

Las constantes en el statusbar

Por otro lado, el listener de selección por punto, se encarga también de actualizar la información existente en la barra de estado de gvSIG. El método que lo hace es el siguiente:

```
private void addConstantsToStatusBar() {
    MDIFrame mF = (MDIFrame) PluginServices.getMainFrame();
    NewStatusBar footerStatusBar = mF.getStatusBar();
    Constants constants = constantManager.getConstants();
    String constantsInfo = getConstantsInfo(constants);
    footerStatusBar.setMessage("constants", constantsInfo);
}
```

La creación de la barra se realiza en el archivo `config.xml` con el siguiente código:

```
<label-set class-name="com.iver.cit.gvsig.project.documents.view.gui.View">
    <label id="units" size="75"/>
    <label id="x" size="120"/>
    <label id="y" size="120"/>
    <label id="4" size="110"/>
    <label id="5" size="110"/>
    <label id="distancearea" size="30"/>
    <label id="projection" size="110"/>
```

```
    <label id="constants" size="140"/> <!-- cuadro de constantes -->
</label-set>
```

Notar que durante las etapas de desarrollo ha bastado con incluir este código en el archivo config.xml, mientras que en producción, ha sido necesario incluir la penúltima línea `<label id="constants" size="140"/> <!-- cuadro de constantes -->` en el propio `config.xml` de la extensión appgvSIG, que es la encargada final de crear y mantener los cuadros en la barra de estado.

Herramientas de digitalización

Resumen

- Las herramientas de digitalización son las contenidas en el proyecto OpenCADTools, una extensión para gvSIG.
- Cada acción en el menú de la extensión extCatastro llama a una de las herramientas de digitalización de las openCADTools.
- Las herramientas de openCADTools proveen de listeners de fin de edición al desarrollador para que pueda procesar la información necesaria al finalizar.

Paquetes java relacionados:

- `es.icarto.gvsig.catastro`: extensiones de digitalización que inicializan las CADTools.
- `es.icarto.gvsig.catastro.wrapperscadtools`: Wrappers para inicializar las CADTools.

En detalle

Se muestran a continuación las herramientas CAD inicializadas por cada una de las extensiones del menú de extCatastro:

- ConstruccionPolygonExtension -> CutPolygonCADTool
- ConstruccionFusionExtension -> SelectionGeometryCADTool
- ConstruccionNewExtension -> InsertAreaCADTool
- ManzanaNewExtension -> InsertAreaCADTool
- PredioDivideExtension -> CutPolygonCADTool
- PredioFusionExtension -> SelectionGeometryCADTool

La operación con cada una de las herramientas puede consultarse en el manual de usuario de las OpenCADTools adjunto.

A nivel implementación, debido a que el lenguaje Java no permite múltiple herencia, se ha hecho lo siguiente:

- Por cada acción del menú, crear una extensión, que hereda de la clase Extension, de gvSIG.
- Cada una de las extensiones mantiene un wrapper sobre la CADTool que necesita. Estos wrappers se pueden encontrar en el paquete `es.icarto.gvsig.catastro.wrapperscadtools`.
- Cada uno de los wrappers, hereda de la extensión que lanza la CADTool (se pueden ver en el proyecto extCAD) y delega la lógica de aplicación en los métodos propios de la extensión padre.

Reglas topológicas y de negocio

Resumen

- La extensión `ActionDispatcher.java` escucha los eventos de fin de edición de las CADTools y, dependiendo de varias variables, decide cuál de las acciones lanzar.
- Cada una de las acciones del menú tiene un evaluador de reglas y de acciones (si es necesario). Ambos se pueden encontrar en el paquete `es.icarto.gvsig.catastro.evaluator`.
- Cada una de las reglas debe implementar la interfaz IRule (si es una regla a cumplir) o IAction (si es una acción a realizar una vez se comprueben todas las anteriores reglas). Además, deben incluirse en el evaluador correspondiente.

Paquetes de interés:

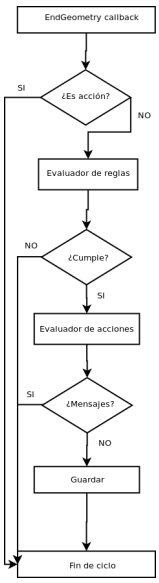
- `es.icarto.catastro`: contiene la extensión *ActionDispatcherExtension.java*`, que implementa el algoritmo de evaluación de reglas y acciones.
- `es.icarto.catastro.evaluator`: contiene los evaluadores de reglas y acciones para cada uno de los items del menú.
- `es.icarto.catastro.evaluator.actions`: contiene las clases de cada una de las acciones a realizar.
- `es.icarto.catastro.evaluator.rules`: contiene las clases de cada una de las reglas a realizar.

En detalle

Una vez el usuario termina la digitalización (creación de una nueva manzana, división de predio, etc), la herramienta de CAD correspondiente lanza un evento de fin de edición. Este evento es capturado por la clase ActionDispatcher y procesado para determinar qué acción es necesario realizar. La decisión se realiza teniendo en cuenta varias variables: la CADTool que lanza el evento y la capa sobre la que se ha realizado.

El flujo para cada uno de los ítems en el menú es similar: una vez entra en el callback se comprueba que es una acción conocida; si es así, se lanza el evaluador de reglas; en el caso de que todas las reglas sean cumplidas se lanza el evaluador de acciones; si todas ellas resultan correctas (no hay mensajes) se guarda y finaliza el ciclo.

Se puede observar el flujo completo del algoritmo en el siguiente gráfico:



Reglas topológicas

Para realizar las comprobaciones topológicas, se usa la librería Java Topology Suite, JTS. En la instalación estándar de gvSIG por defecto tenemos a nuestra disposición la versión 1.9, que ha sido la versión usada.

JTS y gvSIG tienen 2 modelos geométricos diferentes, por lo que es necesario realizar la conversión entre modelos previamente a realizar una operación topológica. Para cualquier operación de este tipo, el flujo ha sido el siguiente:

1. Tomar con gvSIG las geometrías necesarias (realizando filtros, consultas, etc). El resultado es un IGeometry de gvSIG.
2. Convertir las geometrías de gvSIG (IGeometry) a geometrías de JTS (Geometry). La propia interfaz provee un método para realizar esta operación: `ourIGeometry.toJTSGeometry();`
3. Realizar la operación o consulta espacial de entre las disponibles con JTS.
4. Usar el resultado de la operación resultante en nuestros propios algoritmos. Si ésta es una geometría, será necesario convertirla de nuevo a una geometría de gvSIG, lo que puede hacerse con el método `FConverter.jts_to_geometry()`.

Interfaces IRule e IAction

Los evaluadores de acciones y reglas no son más clases que contienen las reglas y las ejecutan iterativamente. Crear una nueva regla/acción es tan sencillo como crear la propia regla/acción y añadirla al evaluador correspondiente.

Es necesario que las reglas implementen la interfaz IRule, que contienen 2 métodos:

- `isObey()`: indica si la operación ha sido realizada con éxito.
- `getMessage()`: que devuelve un mensaje de estado sobre la operación.

Igualmente, las acciones deben implementar la interfaz IAction, que contiene 2 métodos:

- `execute()`: que indica con un booleano si la acción ha terminado correctametne o no.
- `getMessage()`: que devuelve un mensaje de estado sobre la operación

Otras estructuras de interés

Las siguientes estructuras facilitan el trabajo con gvSIG, creando wrappers sobre acciones comunes de la plataforma para que sean más sencillas de utilizar para el desarrollador.

TOCLayerManager.java

Se encarga de mantener el estado de las capas en el TOC (Table of Contents) y provee funciones para consultar su estado y/o cambiarlo.

ToggleEditing.java

Se encarga de la gestión de edición de capas. Contiene métodos para poner en edición la capa, cerrarla y modificar sus contenidos.

Es importante resaltar que, el proceso de edición debe ser tal que así:

```
ToggleEditing te = new ToggleEditing();
te.startEditing(layer);
te.modifyValues();
te.stopEditing(layer, saveOrNotBoolean);
```

Es decir, el proceso de edición requiere poner la capa en edición previamente a poder modificar sus valores y finalmente cerrar la capa, que es el momento en que esos valores se guardarán o no, dependiendo del valor del booleano que se le indique.

Preferences.java

Esta clase contiene variables estáticas del sistema. Algunas de ellas deberían residir en base de datos, de cara a facilitar nuevas instalaciones del sistema en otros entornos.