Islas Canarias
Del 15 al 19 de noviembre de 2021

Generalitat de Catalunya
Institut d'Estadística de Catalunya

# SOBRE LA PRODUCCIÓN DE ESTADÍSTICAS EN MAPAS

**Eduard Suñé**

Subdirecció General de Producció i coordinació.

Àrea de Població i Territori. IDESCAT.

esl@idescat.cat.

**Roser Condal**

Subdirecció General d'Informació i comunicació

Àrea de Tecnologies de la Informació. IDESCAT.

rcondal@idescat.cat.

***Daniel Ibáñez***

Subdirecció General de Producció i coordinació.

Àrea de Població i Territori. IDESCAT.

dibanez@idescat.cat.

# 1. Introducción

Una de las formas más útiles y naturales de observar el comportamiento de una variable estadística en el territorio es mediante uso de mapas y ciertos recursos gráficos.

Un claro ejemplo de estas técnicas de visualización son los denominados mapas de coropletas que permiten ver el comportamiento territorial de una variable estadística de una forma mucho más natural y reveladora de lo que sería su equivalente en forma de tabla.

Con estas técnicas se establece un conjunto de intervalos, se asigna un color a cada uno de ellos y se dibujan los polígonos asociados al territorio, utilizando esos colores para rellenarlos. La imagen que se obtiene, que describe el comportamiento en el territorio de una cierta variable estadística, dependerá claramente de cómo se han establecido ese conjunto de intervalos.

Por otro lado, las variables estadísticas varían su comportamiento en relación con el tiempo y nuestra imagen del fenómeno puede variar si cambiamos la variable temporal. Es decir, nuestro mapa de coropletas se modificará si la referencia temporal cambia.

Por último, nuestra variable de estudio puede observarse en diferentes niveles de detalle territorial ya sean de naturaleza administrativa (sección censal, distrito, municipio, comarca, provincia, etc.) o estadística (zonas definidas como agregaciones de divisiones administrativas o una cierta malla estandarizada de una resolución concreta).

En esta ponencia describiremos con cierto detalle una aplicación general que hemos desarrollado y que permite la visualización de variables estadísticas continuas en el territorio, gestionando los diferentes tipos de dimensiones que hemos descrito anteriormente:

- Conjuntos de variables estadísticas a representar
- Nivel geográfico a representar
- Variable temporal
- Métodos de cálculo del conjunto de intervalos

El desarrollo de esta aplicación se ha realizado utilizando software libre (Geoserver[1], PosGIS[2], OpenLayers[3], Tomcat[4]), utilizando los protocolos WMS[5] de la OGC para la parte GIS y ha sido escrita utilizando exclusivamente Java como lenguaje de programación, tanto en la parte servidor como en la parte cliente, gracias al uso de Google Web Toolkit[6].

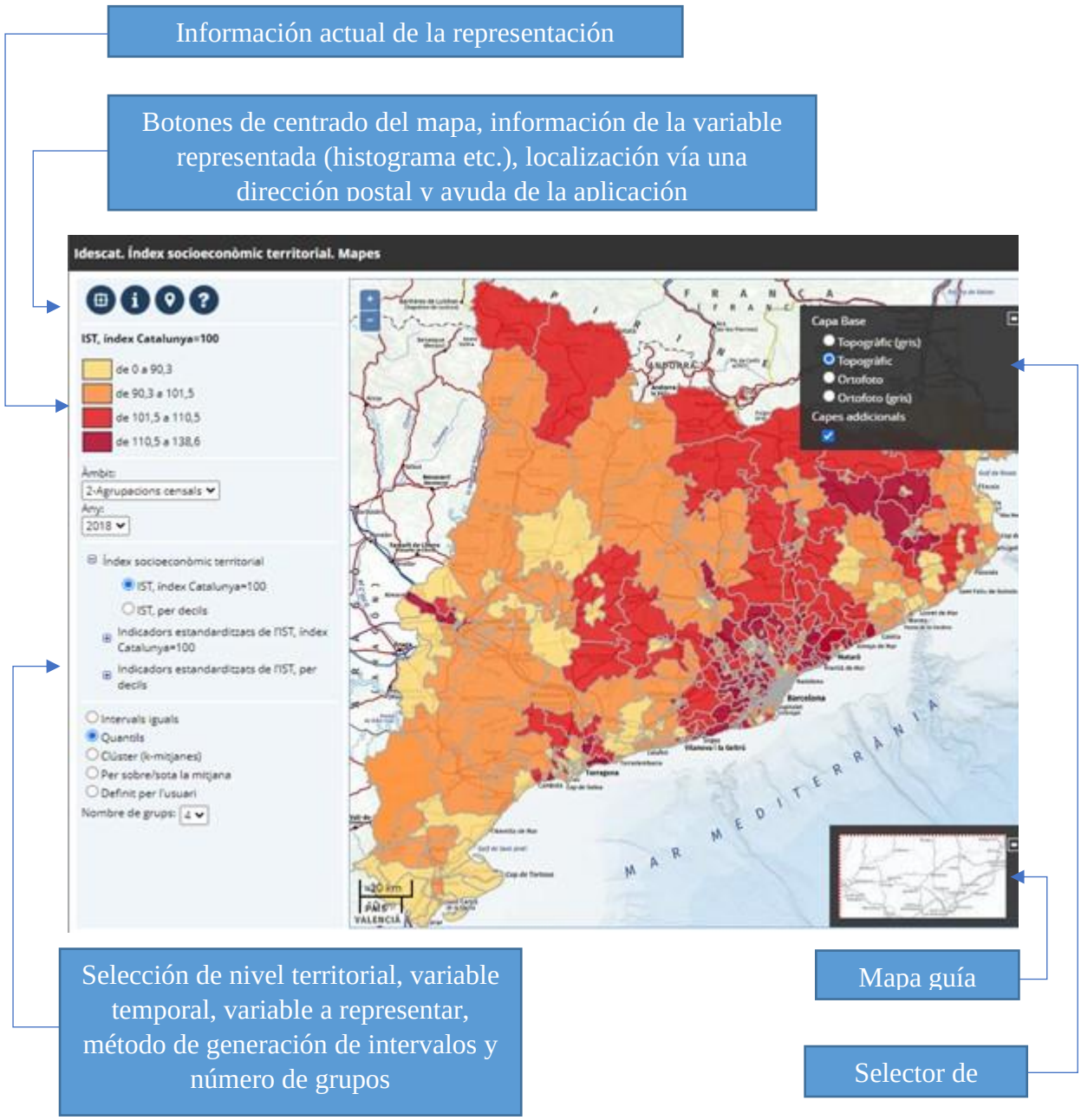
A nuestro entender, el uso exclusivo de Java en todo el desarrollo tiene muchas ventajas como son la seguridad, la modularidad y la extensibilidad de la aplicación, comparada con el típico desarrollo JavaScript en el cliente y Java en el servidor.

## 2. **Objetivos**

Presentamos una lista de requerimientos que han sido implementados en la aplicación para que el lector se haga una idea de cuál es la funcionalidad de la aplicación. Más adelante se irán describiendo las estructuras de datos y métodos implementados para alcanzar cada uno de ellos.

- ✓ Las variables para representar se ordenan jerárquicamente según una temática
- ✓ Para cada variable el usuario puede escoger un nivel territorial determinado
- ✓ Para cada variable y nivel, el usuario puede escoger una referencia temporal
- ✓ El usuario puede escoger alguno de los métodos de creación de grupos siguientes: intervalos iguales, división en cuantiles, clúster utilizando el algoritmo KMeans [7], por encima y por debajo de la media y definición de intervalos por parte del usuario
- ✓ El usuario puede seleccionar el número de grupos, hasta un máximo de siete
- ✓ Al seleccionar un elemento territorial se informa al usuario de la posición en el histograma de ese elemento, la variación temporal de la variable seleccionada, los valores asociados a elementos territoriales superiores al escogido (si estamos a nivel de sección, el valor a nivel de distrito, municipio, comarca y provincia) y, por último, el conjunto completo de valores para ese elemento territorial de las variables que en la jerarquía de variables tienen relación
- ✓ El usuario puede escoger la gama de colores asociado al mapa de entre tres paletas diferentes y el nivel de opacidad en el relleno del mapa
- ✓ El usuario puede escoger diferentes capas base, mover, hacer zoom y disponer de un mapa guía
- ✓ Para la variable escogida, el usuario puede informarse de: aspecto de su histograma, de los parámetros de tendencia central y de dispersión, encontrar elementos atípicos en la distribución (e ir a uno de ellos en el mapa si lo selecciona), la descripción de la variable escogida y de la fuente asociada a los datos
- ✓ Localizar mediante geocodificación una dirección postal en el mapa

Así pues, el aspecto general de la aplicación debería ser:




Al seleccionar un elemento territorial, y también al clicar sobre cualquiera de los botones, deben aparecer una serie de diálogos modales con la información relativa a cada una de las utilidades descritas en los requerimientos de la aplicación.

El hecho de que en los requerimientos aparezca la visualización del histograma relativo a la variable seleccionada para cada referencia temporal o, en el caso de selección de un elemento territorial, también aparezca el histograma y el valor asociado a ese elemento, impone la restricción de que estos histogramas deben ser previamente calculados y almacenados en la base de datos.

En efecto, aunque el número de microdatos (es decir, la cardinalidad asociada a un cierto nivel territorial) pueda ser relativamente pequeño (unos 940 para municipios o unas 5.000 para secciones censales), no lo sería en general.

Por ejemplo, para un grid multiresolución [8] de precisión máxima de 62,5 metros, el número de elementos sería tan elevado que el cálculo del histograma bajo petición sería poco eficiente e inviable para una aplicación en un entorno web.

Además, en los requerimientos de creación de los grupos se plantea la necesidad de que sean, además de intervalos iguales, en base a cuantiles y, por lo tanto, es necesaria una lectura ordenada de los microdatos para su cálculo, lectura que por las razones expuestas no sería eficiente realizarla bajo petición del usuario.

Lo mismo ocurre en el caso del cálculo de grupos mediante técnicas clúster: aunque el método empleado sea muy eficiente (KMeans [9]), es de nuevo necesario tener los valores previamente calculados.

Así pues, es necesario que los datos de partida sean estables y que los cálculos tanto del histograma como de los intervalos asociados a los diferentes métodos sean realizados previamente y almacenados en la base de datos. Como veremos más adelante, es posible automatizar esos cálculos en función de la información residente en la meta base.

Salvo esta consideración y el hecho de que la variable a representar sea numérica y que el tipo de representación sea el de mapas de coropletas, no existen más condiciones previas en cuanto a los datos, siendo la aplicación lo suficientemente general como para gestionarlos adecuadamente.

## 3. **Metodología**

Como es usual en aplicaciones Web, la arquitectura general se basa en un esquema en tres capas: cliente, servidor web y base de datos.

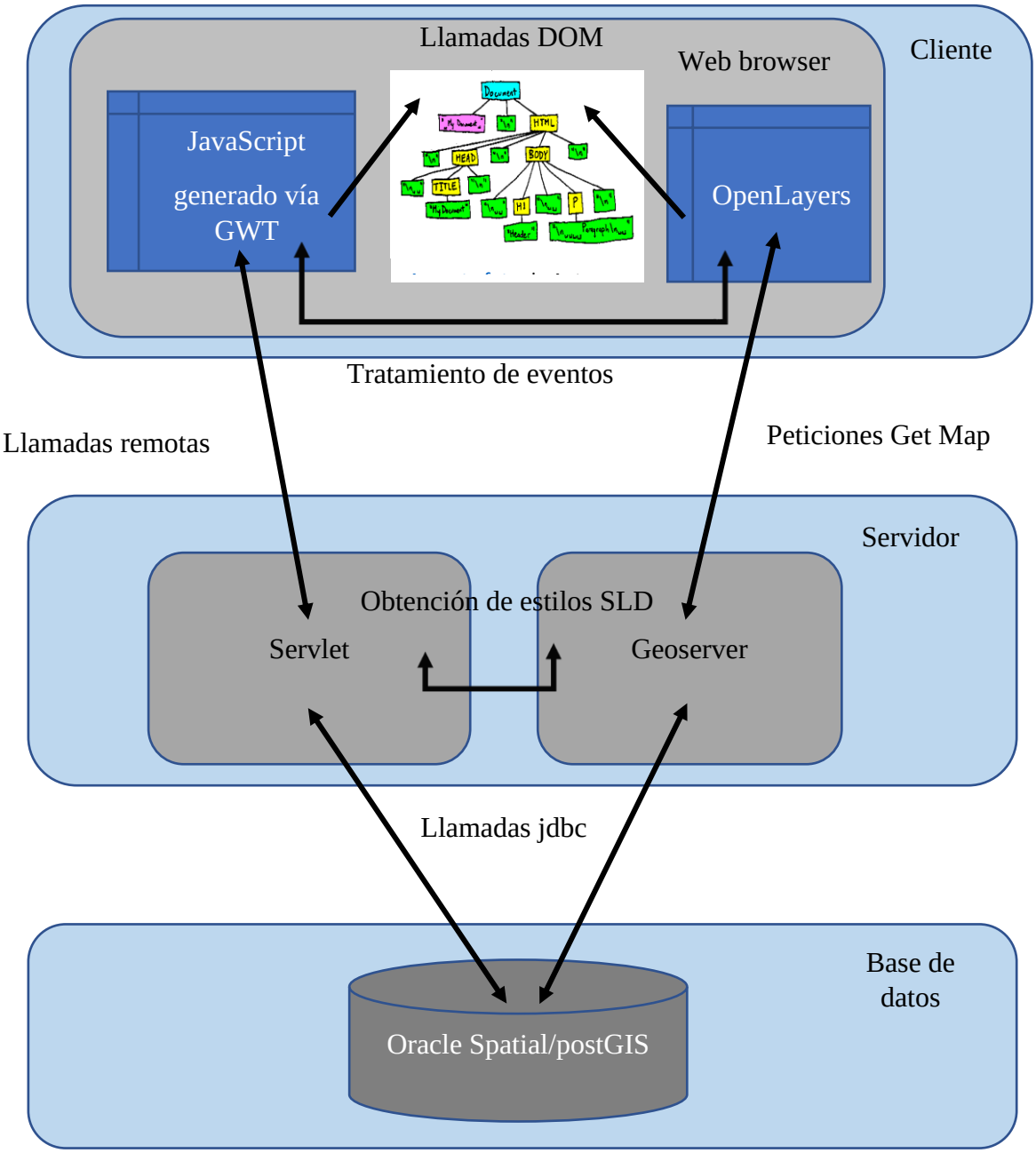
A nivel de cliente se ha utilizado como elemento GIS el conocido software libre OpenLayers, como elementos de representación gráfica el API gráfico de Google y el resto, elementos HTML controlados vía DOM [10] mediante un código JavaScript que, como se verá más adelante, se ha generado automáticamente mediante el framework Google Web Toolkit.

A nivel de servidor destacan dos componentes: el componente GIS (Geoserver) y el componente que gestiona las peticiones “http” ordinarias (un servlet desarrollado *ad hoc*). Estos dos componentes se ejecutan dentro del servidor de aplicaciones Tomcat.

En cuanto a la base de datos, se pueden utilizar indistintamente postGIS-postgreSQL u Oracle Spatial [11] ya que en el código de servidor existen diferentes implementaciones según cual sea el servidor utilizado (de hecho, se implementó primero para postGIS y se añadió posteriormente la implementación que Oracle Spatial precisaba).

Esto es necesario ya que, aunque tanto postGIS como Oracle Spatial cumplen con las especificaciones SFSQL [12], las implementaciones de cada gestor son radicalmente diferentes.

El esquema general de la arquitectura de la aplicación es el siguiente:




A nivel de cliente, mediante un código JavaScript generado vía GWT, se gestionan los datos recibidos mediante llamadas AJAX al servlet y que consisten principalmente en unas estructuras que contienen información sobre las variables a representar, sus posibles niveles geográficos y los valores de la variable temporal.

También se obtiene para cada conjunto variable, año, tipo de ámbito y tipo y número de grupos, la información relativa al histograma e intervalos. La comunicación entre el

servlet y el cliente se realiza de forma asíncrona, pero gracias a la utilización de GWT se reduce notablemente la complejidad del código (este extremo se tratará con más detalle más adelante).

El GUI de la aplicación se actualiza mediante llamadas DOM en función de los datos recibidos y de los eventos producidos como consecuencia de la interacción del usuario.

Una vez realizada una selección de los datos primarios (variable, tipo de ámbito y año) y de la forma de representación (método de creación de grupos y número de grupos), se realizan una serie de llamadas al API de OpenLayers para que éste realice las llamadas correspondientes a Geoserver para obtener el mosaico de imágenes (siguiendo el protocolo WMS) que finalmente conforman el mapa. En el intermedio, Geoserver realiza las llamadas pertinentes al servlet para obtener la información de estilo a aplicar siguiendo las especificaciones de SLD.

Este es el ciclo de eventos que se van produciendo mientras el usuario va cambiando los parámetros básicos de la aplicación.

Existen otros más referidos a la selección de un elemento territorial o al del conjunto de botones de utilidades (información de la variable, geocodificación, etc.), pero el comportamiento general es el mismo: llamada asíncrona al servlet para obtener ciertos datos y manipulación del GUI vía DOM por parte del código JavaScript generado por el compilador GWT.

Hay que tener en cuenta que Geoserver, un servlet que reside en el servidor de aplicaciones Tomcat, debe configurarse correctamente para que pueda representar un conjunto de capas asociadas a una cierta fuente de datos. Además, debe existir una relación clara entre esa configuración y la que correspondería a la meta base utilizada por la aplicación que estamos describiendo.

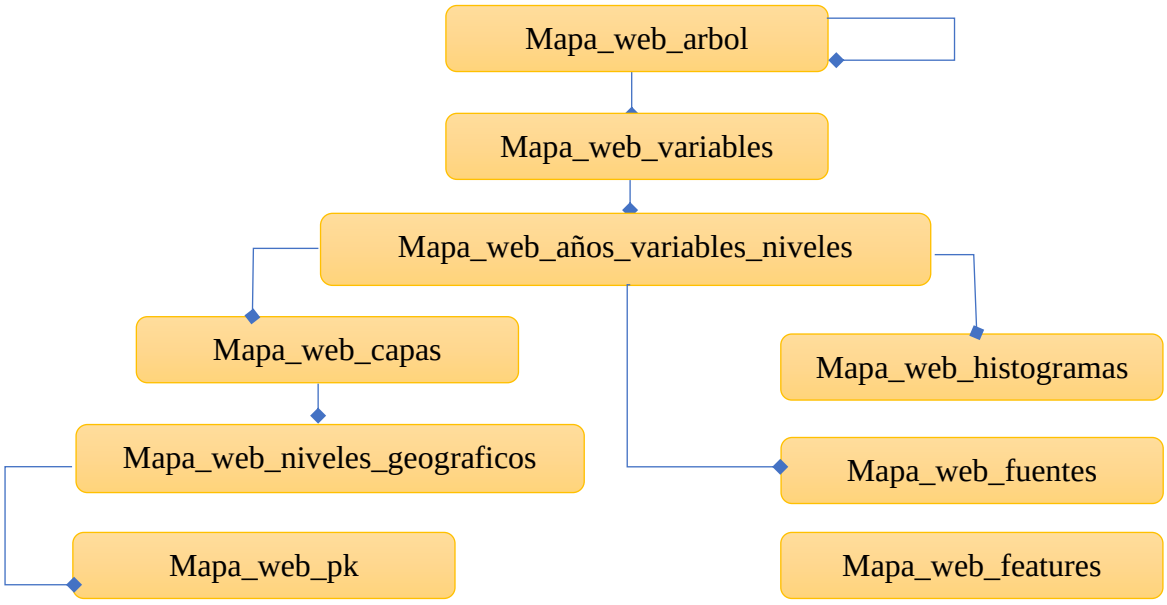
Los datos a representar, que son vistas parametrizadas relativas a un cierto espacio de trabajo (de una cierta fuente de datos) en Geoserver, se corresponden a una cierta vista o tabla en nuestra base de datos.

Esta vista contiene la serie temporal de un conjunto de variables estadísticas para un cierto nivel geográfico y dado que Geoserver va a representar cada vez los valores relativos a una determinada fecha, es necesario que nuestra vista contenga índices definidos para la variable temporal. Además, es necesario definir índices tipo RTree [13] para que el acceso a las geometrías sea eficiente.

A nivel de datos, nuestra base de datos sólo contiene este tipo de vistas, de forma que una vez Geoserver está configurado adecuadamente, será capaz de realizar la representación de nuestra capa en función de un cierto estilo que está asociado a la variable escogida por el usuario. Por ejemplo, para una serie de indicadores demográficos a nivel de sección censal, tendríamos la tabla SEC_DEM_GEOM, tabla (o vista) que debe contener las series temporales y las geometrías relativas a su nivel geográfico:

| Columna | Tipo de dato | Descripción |
|---|---|---|
| PRO | VARCHAR2(2) | Código de provincia |
| MUN | VARCHAR2(3) | Código de municipio |
| DIS | VARCHAR2(2) | Código de distrito |
| SEC | VARCHAR2(3) | Código de sección |
| ANO | VARCHAR2(4) | Año de referencia |
| POB | NUMBER | Población |
| PERC_HOMES | NUMBER | % hombres |
| PERC_DONES | NUMBER | % mujeres |
| RELACIO_MASCULINITAT | NUMBER | Relación de masculinidad |
| PERC_EDAT_0_14 | NUMBER | % población de 0 a 14 años |
| PERC_EDAT_15_64 | NUMBER | % población de 15 a 64 años |
| PERC_EDAT_65_84 | NUMBER | % población de 65 a 84 años |
| PERC_EDAT_85_I_MES | NUMBER | % población de 85 o más años |
| DEPENDENCIA_JUVENIL | NUMBER | Índice de dependencia juvenil |
| DEPENDENCIA_SENIL | NUMBER | índice de dependencia senil |
| PERC_NASC_CAT | NUMBER | % población nacida en Catalunya |
| PERC_NASC_ESP | NUMBER | % población nacida resto estado |
| PERC_NASC_EST | NUMBER | % población nacida en el extranjero |
| PERC_NACIO_ESPA | NUMBER | % población española |
| PERC_NACIO_ESTRAN | NUMBER | % población extranjera |
| GEOM | SDO_GEOMETRY | Geometría (SRID 25831) |

Para que la aplicación pueda gestionar adecuadamente los datos a publicar, es necesario definir un conjunto de tablas en la base de datos que describa precisamente su propio contenido, es decir, disponer de una meta base que permita cumplir con los requerimientos descritos al principio de este documento. El esquema entidad relación de la meta base es el siguiente:



La tabla ***Mapa_web_arbol*** contiene la información del árbol de variables y por esta razón existe la relación reflexiva entre sus elementos.

La tabla ***Mapa_web_variable*** contiene los descriptores de las variables (títulos que aparecerán en la aplicación) así como los nombres internos de las columnas de las tablas de datos.

La tabla ***Mapa_web_años_variables_niveles*** contiene las combinaciones existentes de variable, ámbitos y años y un atributo de visibilidad.

La tabla ***Mapa_web_capa*** establece la tabla de datos relativa a cada capa de Geoserver.

La tabla ***Mapa_web_histogramas*** contiene, para cada una de las combinaciones variable, nivel territorial y año, un objeto binario que es la serialización en la base de datos del objeto java Histograma (del cual haremos una descripción en el siguiente apartado). Como su nombre indica, contiene toda la información relativa al histograma, valores de tendencia central y de dispersión, información relativa a los intervalos para cada tipo de cálculo (intervalos iguales, cuantiles, cluster KMeans etc).

Este objeto se calcula mediante consultas a la tabla de datos si no existe en la tabla ***Mapa_web_histogramas*** y se serializa en el momento de inicialización del servlet, de tal forma que siempre está disponible dicha información una vez la aplicación ha arrancado. El cálculo se realiza para todas las combinaciones existentes en la tabla ***Mapa_web_años_variables_niveles*** y se carga en una tabla hash para su lectura por parte de los clientes (mediante llamadas remotas asíncronas).

El resto de las tablas contienen información necesaria para el funcionamiento adecuado de la aplicación. Así, por ejemplo, la tabla ***Mapa_web_features*** contiene información sobre la activación o no de ciertos botones en función de ciertos perfiles de usuarios (por ejemplo, para activar o no la búsqueda de valores atípicos).

Con toda esta información, para poder difundir una nueva información se deben realizar, obviamente, una serie de pasos:

- Crear la información a difundir (la serie temporal con los indicadores)
- Configurar Geoserver para poder acceder a esos datos vía vistas parametrizadas
- Realizar las altas adecuadas en la metabase para la difusión de esos datos

Una vez culminados estos pasos, la aplicación mostrará la información cumpliendo con cada uno de los requisitos que al principio del documento hemos descrito.

Google Web Toolkit es un framework que permite el desarrollo en lenguaje Java de aplicaciones Web. Consiste en un API que implementa las manipulaciones de páginas web vía DOM para construir la interfaz gráfica y tratamiento de eventos. Además, simplifica las llamadas asíncronas AJAX mediante un protocolo que recuerda a Java RMI [14]. Finalmente, dispone de un compilador que, una vez ha actuado el compilador Java, convierte el código Java en JavaScript que es el que finalmente se ejecuta en el cliente (el navegador).

El hecho de que el desarrollo se realice exclusivamente en Java tiene una serie de ventajas derivadas del tipo de lenguaje y que no es posible encontrarlas en un lenguaje interpretado como JavaScript.

Otra de las ventajas que presenta es la simplificación y seguridad que se obtiene con su solución a las llamadas tipo AJAX.

Veamos, por ejemplo, la clase Histograma que es un elemento de vital importancia en la aplicación:

***public class Histograma***

***implements java.io.Serializable,com.google.gwt.user.client.rpc.IsSerializable***

*{……..}*

Esta clase contiene una serie de campos que permiten construir el histograma de una cierta variable, para un cierto tipo de nivel territorial y año. También contiene una serie de métodos que permiten la obtención de valores concretos de un objeto (geters), llenado de valores (seters) o, por ejemplo, el cálculo de los límites asociados a los cuantiles.

Pues bien, los objectos de esta clase se crean y serializan en el servidor, pero puede accederse desde el cliente ya que implementa la interfaz ***com.google.gwt.user.client.rpc.IsSerializable*** que es la única condición para que pueda transmitirse como resultado o argumento de una llamada remota.

Así, por ejemplo, en el lado del servidor, nuestro servlet **(SLDGenerator**) dispone de diferentes métodos que pueden llamarse desde el cliente. En su definición observamos:

***public class SLDGenerator extends***
***com.google.gwt.user.server.rpc.RemoteServiceServlet implements***
***org.idescat.CensusMaps.client.SLDGeneratorRemote***

*{……..}*

La interfaz ***org.idescat.CensusMaps.client.SLDGeneratorRemote*** está definida como:

***public interface SLDGeneratorRemote extends***
***com.google.gwt.user.client.rpc.RemoteService*** *{*

*public Histograma* ***getHistograma****(String variable, int id ,String taulaBD,String ambit,String any);*

*public Metadades* ***getMetadade****s(int root,boolean simulaFora);*

*public String[]* ***getDscMetodes****();*

*public String* ***getDscFont****(String idFont);*

*public String[]* ***getDscVariableFont****(String nom_var,String capa);*

*public String* ***getInfoPunt****(double x,double y);*

*public String[]* ***getInfoPuntAllDades****(double x,double y,String[][] varTaules);*

*public ResultInfoPunt **getInfoPuntAllDades**(ArgumentsInfoPunt arg ,boolean simulaFora);*

*public ResultInfoPunt **getInfoPuntAllDades**(ArgumentsInfoElementSeleccionat arg, boolean simulaFora);*

*public String[] **getMunicipis**();*

*public PuntGeocodificat[] **getAdress(**String carrer,int numero,String municipi);*

*public ResultInfoMesSemblants **getInfoPuntMesSemblants**(arg,double valor,int limit);*

*public.Outliers **getOutliers**(String taula,String variable,String any,double mitjana,double sttdev);*

*public ConfiguradorFeatures **getConfiguracioFeatures**(int perfil);}*

Aquí podemos ver que el primer método **getHistograma** devuelve una instancia del objeto **Histograma** pasando como argumentos los valores relativos a la variable, el nivel territorial y el año. A partir de ese momento, el cliente ya puede disponer de los datos y métodos de ese objeto.

Está claro que, tanto a nivel de servidor como de cliente, no se está ejecutando el mismo código ya que todo lo que forma parte del cliente se va a traducir a JavaScript en el momento en que el compilador GWT actúe, pero a un nivel superior de abstracción podemos decir que es el mismo objeto. El mecanismo de llamada remota GWT es siempre el mismo, por ejemplo:

```
infoVarAsync.getHistograma(nomvar, idvar, taulaBD, ambit, any, new com.google.gwt.user.client.rpc.AsyncCallback() {
        public void onFailure(Throwable caught) {
           DialogError dg=new DialogError();
           return;}
             public void onSuccess(Object result) {
             if(result == null) {
             inf.histograma=null;
              DialogError dg=new DialogError("MEP-004"); }
             else{
              inf.histograma=(org.idescat.CensusMaps.client.Histograma)result;
              meta.putHistograma(inf.histograma,nomvar,ambit,any);}
        }
 }
```

aquí la variable **infoVarAsync** está definida como:

***infoVarAsync=(org.idescat.CensusMaps.client.SLDGeneratorRemoteAsync)GWT.create(org.idescat.CensusMaps.client.SLDGeneratorRemote.class);***

es decir, es una variable que encapsula la implementación asíncrona de la interfaz remota y que nos permite realizar las llamadas remotas definidas en **SLDGeneratorRemote**.

En cuanto a la generación y manipulación de la interfaz gráfica de la aplicación, GWT contiene un extenso API (que recuerda a Swing) y que internamente, una vez compilado a JavaScript, realiza manipulaciones DOM del documento.

Existen implementaciones GWT de muchísimos recursos, entre ellos de OpenLayers [15], de tal forma que el código correspondiente a la parte GIS del cliente ha podido realizarse sin tener que apartarse del framework GWT, es decir mediante código Java que posteriormente se traducirá a JavaScript gracias al compilador GWT (hay que comentar que siempre cabe la posibilidad de insertar en el código java llamadas nativas de JavaScript, aunque tal posibilidad no ha sido necesaria en nuestro caso).

Los métodos que implementa el servlet **SLDGenerator** definidos en la interfaz **SLDGeneratorRemote** son necesarios para alcanzar todos y cada uno de los requisitos descritos:

- ***Histograma getHistograma(String variable,int id_variable,String taulaBD,String ambit,String any);***
  - Obtención del histograma para una variable, nivel territorial y año.
- ***Metadades getMetadades(int root,boolean simulaFora);***
  - Obtención de los metadatos partiendo de un nodo padre del árbol de variables. Permite obtener las estructuras de datos completas necesarias para que el usuario pueda seleccionar variables, ámbitos, años, método de generación de intervalos etc. Este método realiza una lectura completa de toda la metabase partiendo de un nodo padre del árbol de variables.
- ***public String[] getDscMetodes();***
  - Obtención de la descripción de los métodos de representación.
- ***public String getDscFont(String idFont);***
  - Obtención de la descripción de una fuente.
- ***public String[] getDscVariableFont(String nom_var,String capa);***
  - Obtención de la descripción de una variable para una determinada capa.
- ***public ResultInfoPunt getInfoPuntAllDades(org.idescat.CensusMaps.client.ArgumentsInfoElementSeleccionat arg,boolean simulaFora);***
  - Obtención de información relativo al elemento seleccionado en el mapa.

- ***public String[] getMunicipis();***
    - Obtención del array de municipios (necesario para la geocodificación posterior).
- ***public PuntGeocodificat[] getAdress(String carrer,int numero,String municipi);***
    - Punto obtenido mediante geocodificación de una dirección postal.
- ***public ResultInfoMesSemblants getInfoPuntMesSemblants(org.idescat.CensusMaps.client.ArgumentsInfoPunt arg,double valor,int limit);***
    - Búsqueda de elementos de un cierto nivel territorial similares al escogido en relación con el valor de una variable.
- ***public org.idescat.CensusMaps.client.Outliers getOutliers(String taula,String variable,String any,double mitjana,double sttdev);***
    - Búsqueda de valores atípicos en la distribución utilizando como criterio Z-score.
- ***public org.idescat.CensusMaps.client.ConfiguradorFeatures getConfiguracioFeatures(int perfil);***
    - Configuración de la aplicación según el perfil.

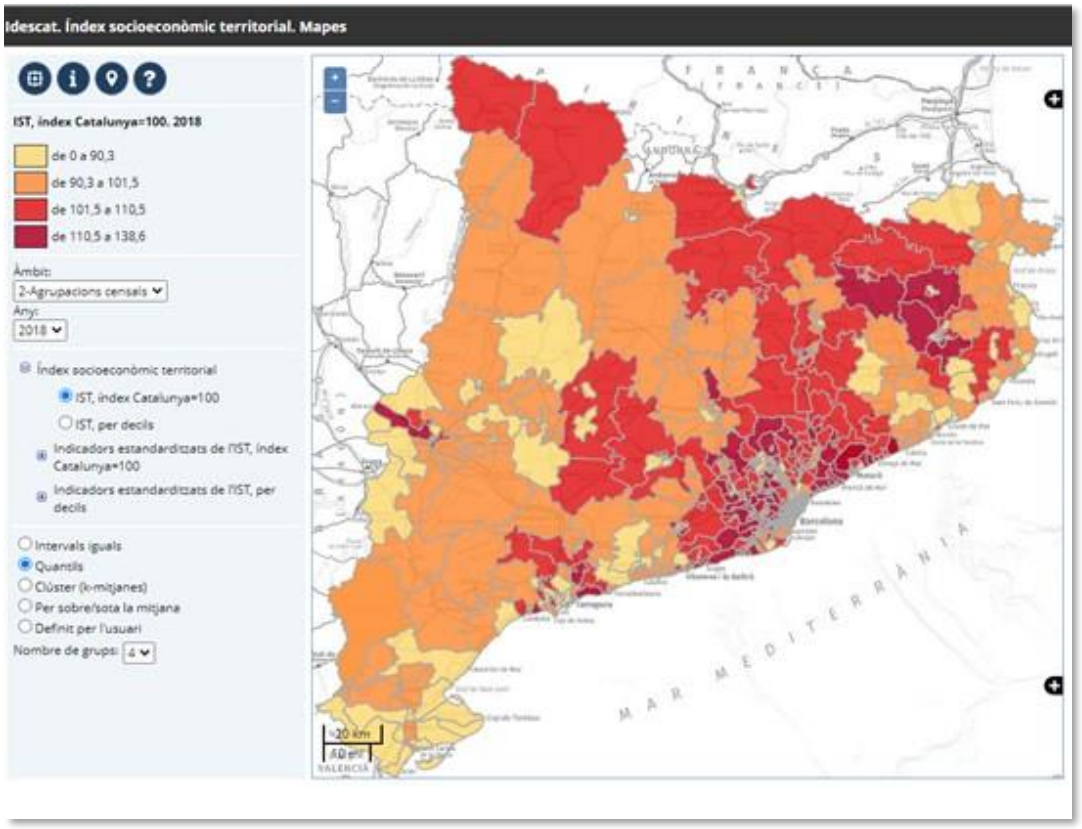
## 4. **Resultados**

Como ya hemos comentado el código que realmente se ejecuta en el cliente es código JavaScript que se ha generado mediante una compilación especial a partir de código Java.

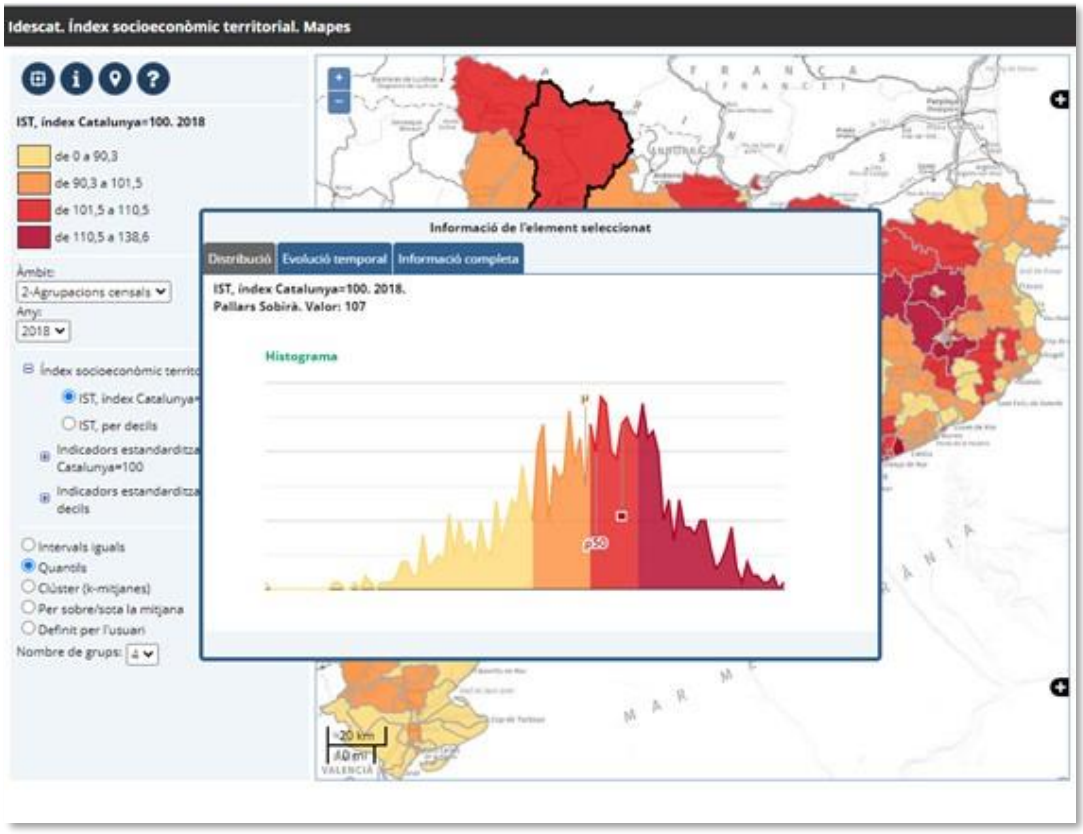
Cuando la página se carga, se ejecuta un método estándar (**OnModuleLoad**) que es uno de los definidos en GWT para la interfaz **EntryPoint** y que implementa nuestra clase principal. Al ejecutarse se realizan las siguientes acciones:

- ✓ Llamada remota a ***getConfiguracioFeatures*** para configurar la aplicación
- ✓ Llamada remota a **getMetadades** para obtener toda la información de la metabase
- ✓ Llamada remota a **getHistograma** para la variable, tipo de ámbitos y año por defecto
- ✓ Llamada a **putGUI** que crea toda la interfaz gráfica de la aplicación

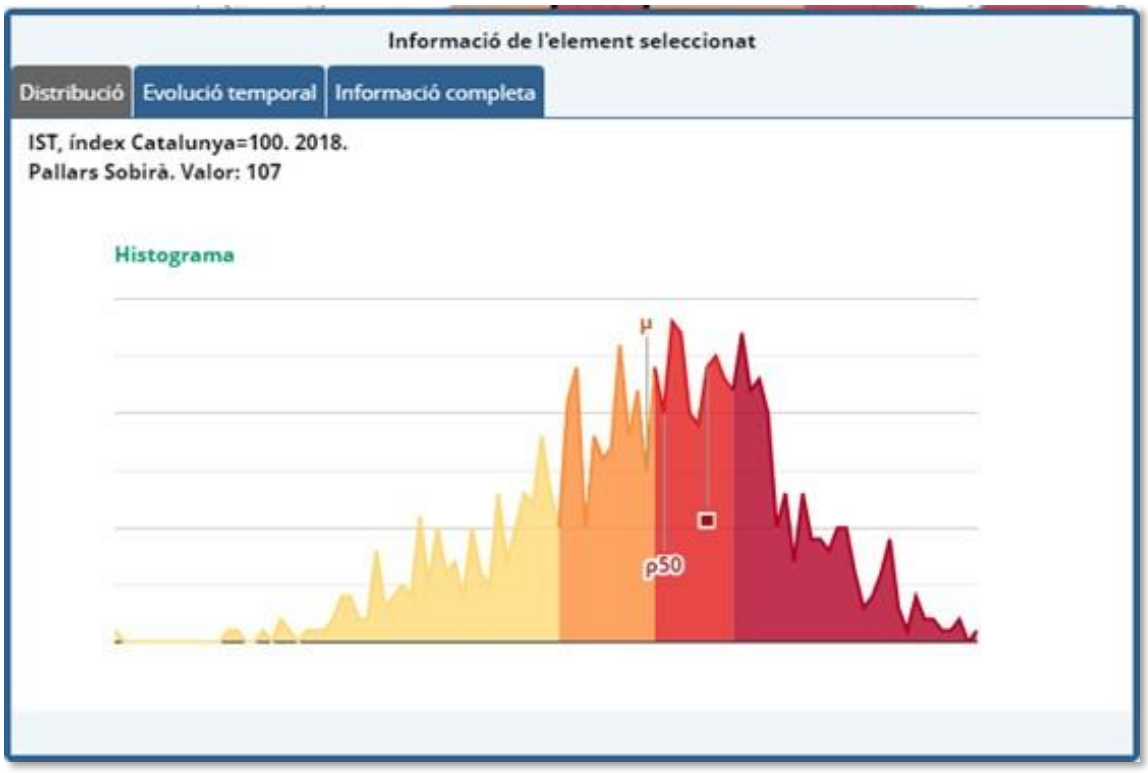
A partir de este momento, la aplicación está a la espera de que se generen eventos como consecuencia de las acciones del usuario y tiene el siguiente aspecto:



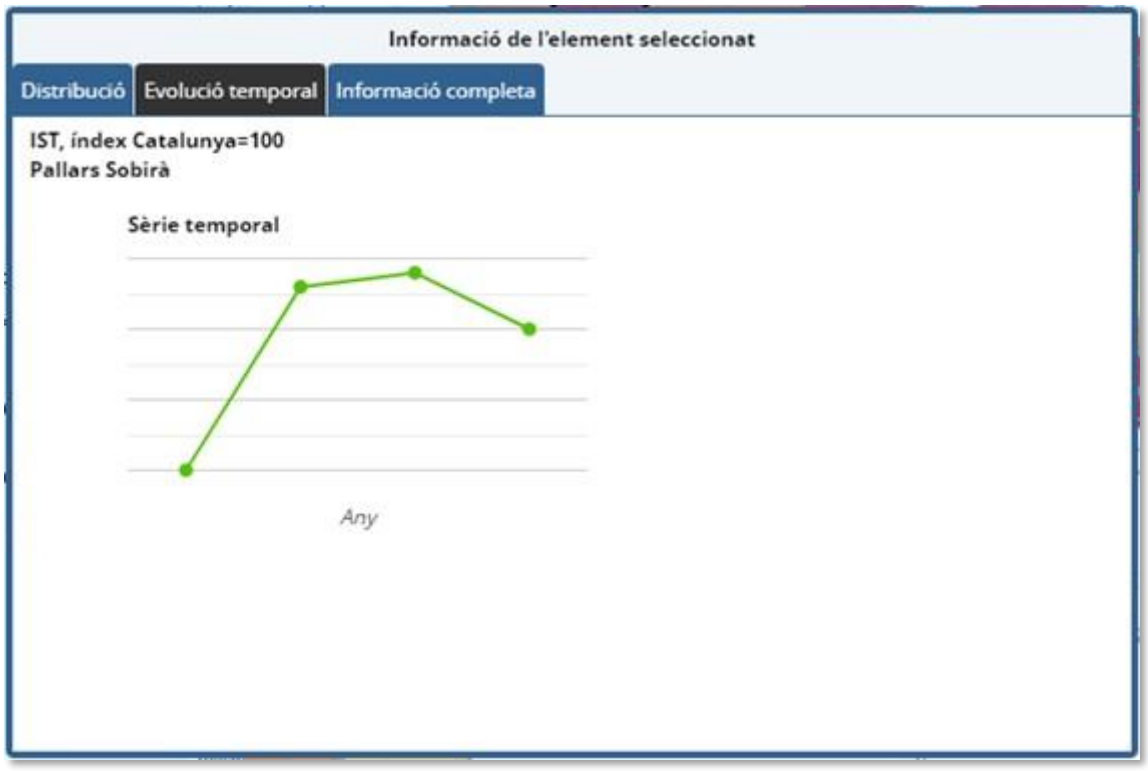

Si el usuario selecciona un elemento territorial, se disparan una serie de eventos (definidos en el API OpenLayers) que la aplicación trata, obteniéndose información de ese elemento territorial, mediante un dialogo modal:



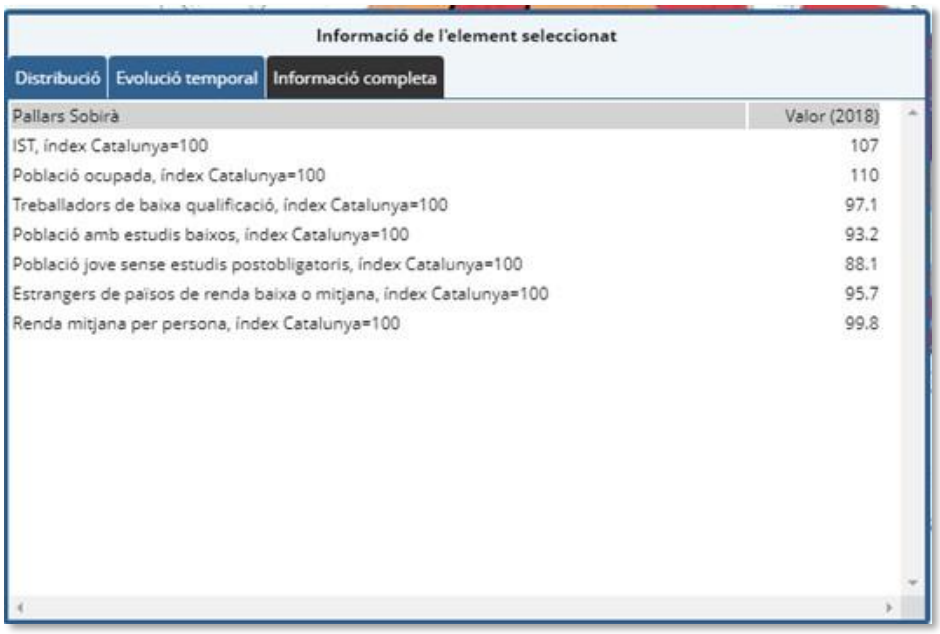
Esta información consiste en la posición de ese elemento en el histograma de la variable:



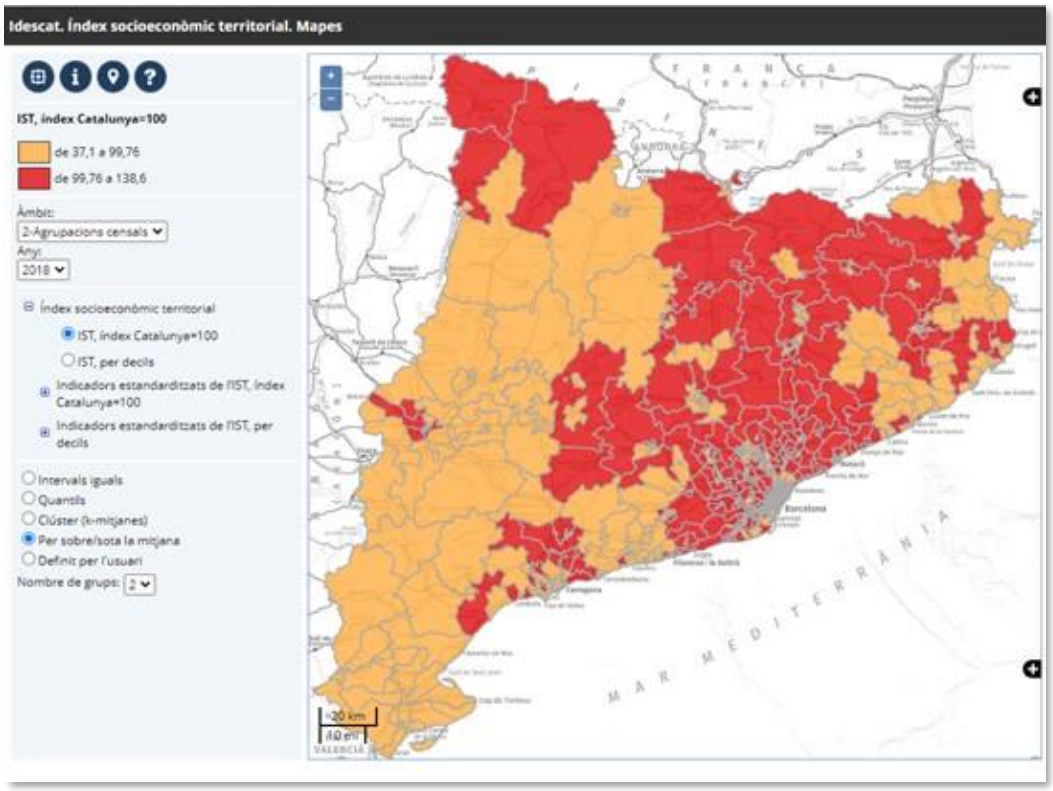

en la evolución temporal de la variable para ese ámbito:



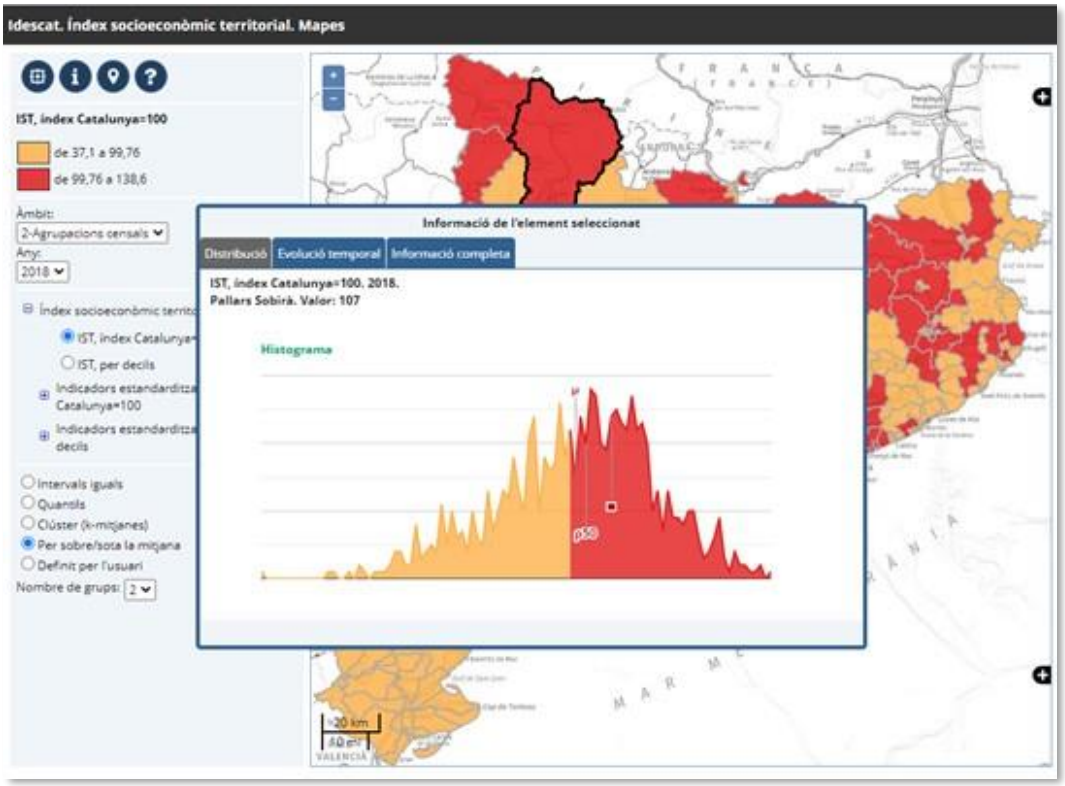
y en la información completa, relativa a la raíz del árbol de variables, para ese ámbito:



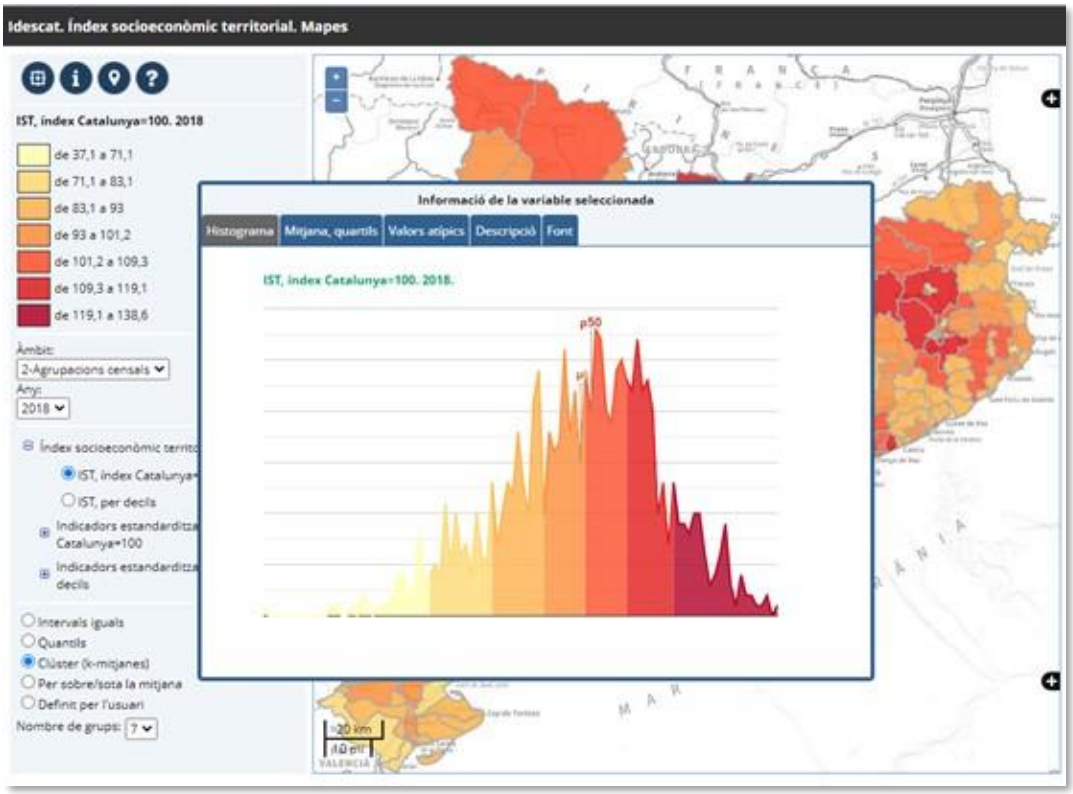

Obsérvese que, en el histograma, los colores de relleno están sincronizados con los colores del mapa de coropletas y que queda señalada la posición del valor del ámbito junto con la media y la mediana de la variable escogida. Además, si el usuario decide cambiar el método de presentación, se producen y tratan los eventos de forma adecuada, obteniéndose un nuevo mapa de coropletas:



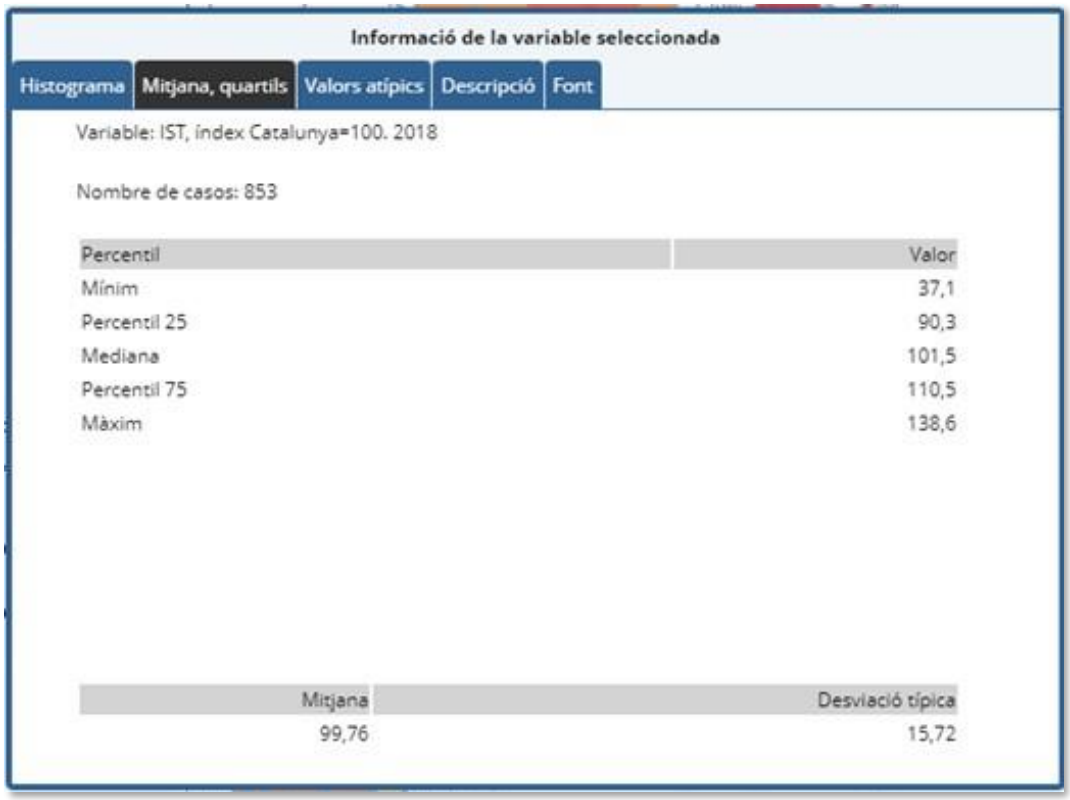
y al clicar de nuevo en algún ámbito, la información representa los nuevos parámetros de consulta:



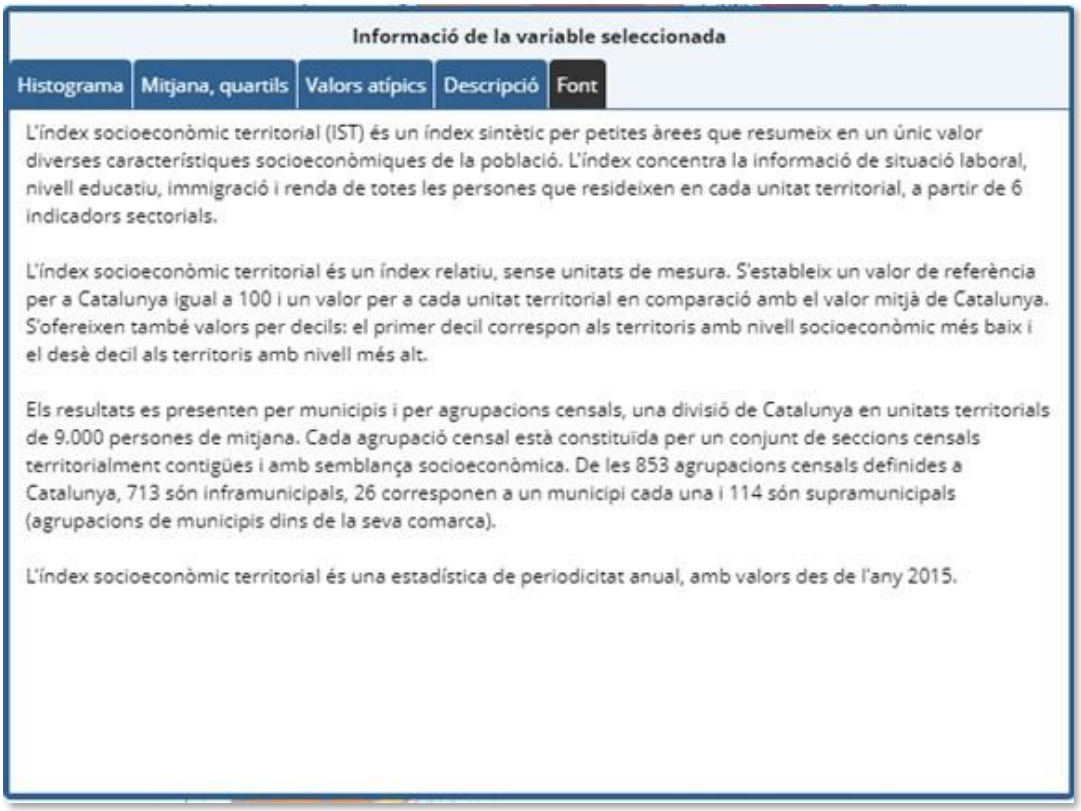

Al clicar en el botón de información de la variable aparece otro conjunto de informaciones:



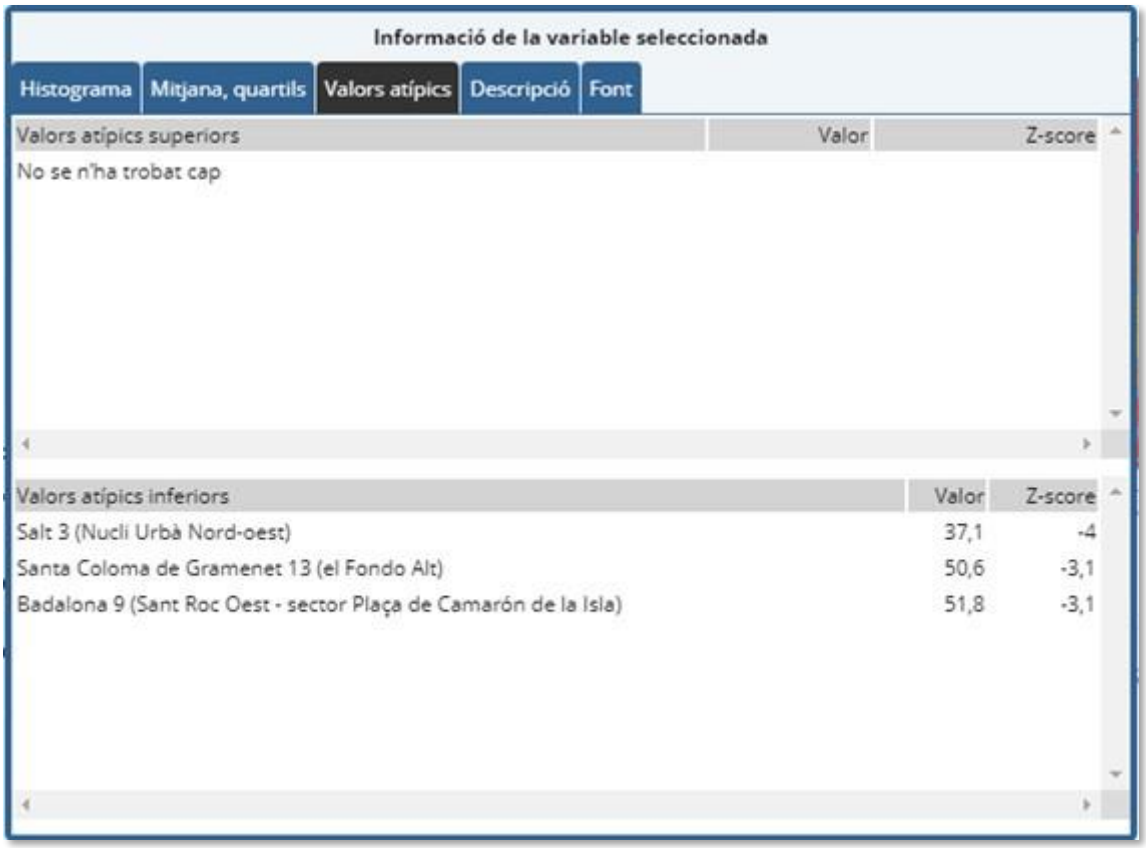
En este caso obtendríamos el histograma de la variable, con los colores e intervalos que haya seleccionado el usuario (en este caso un clúster KMeans con siete grupos), los valores de tendencia central y dispersión:



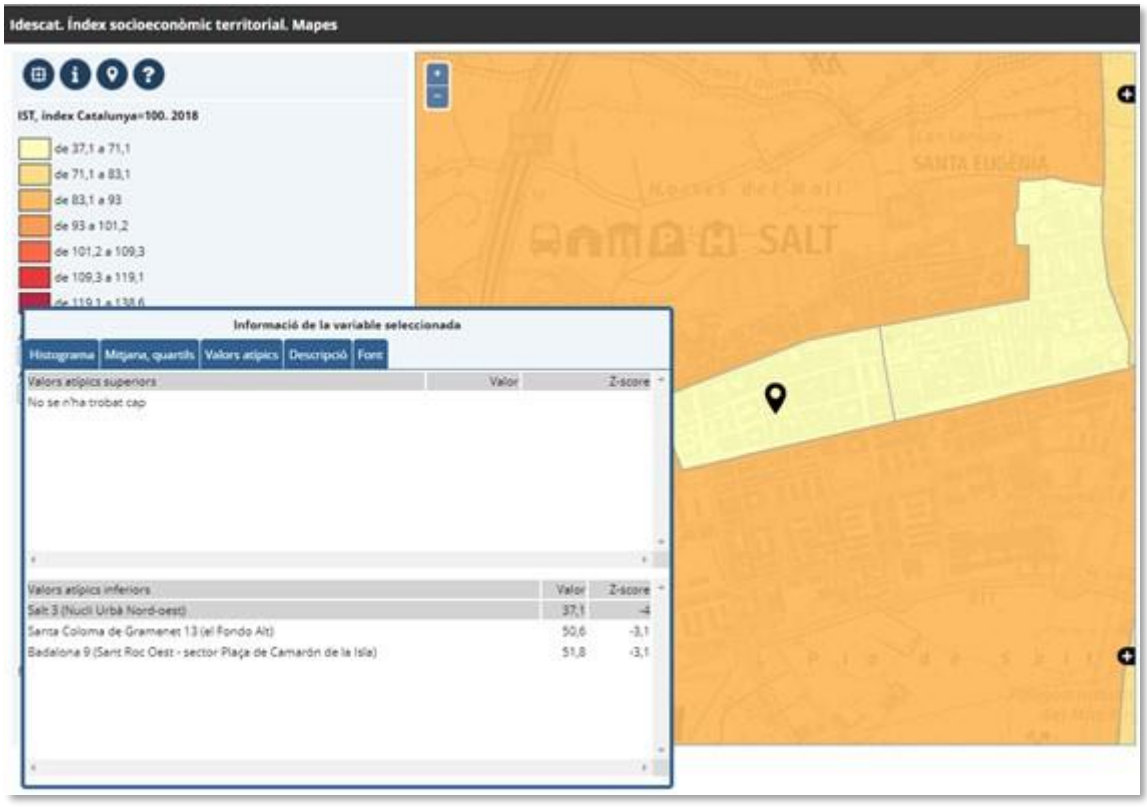

la descripción de la variable y la fuente asociada:



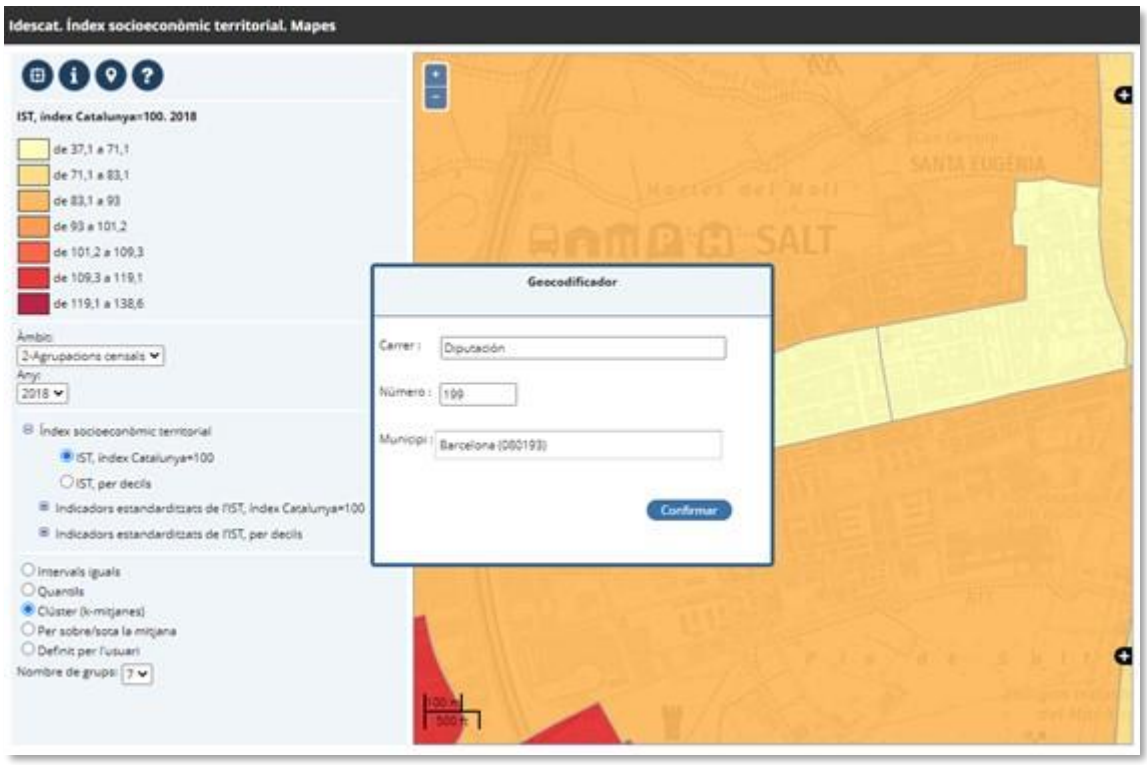
y, finalmente, los valores atípicos (calculados según los valores extremos de los z-scores):



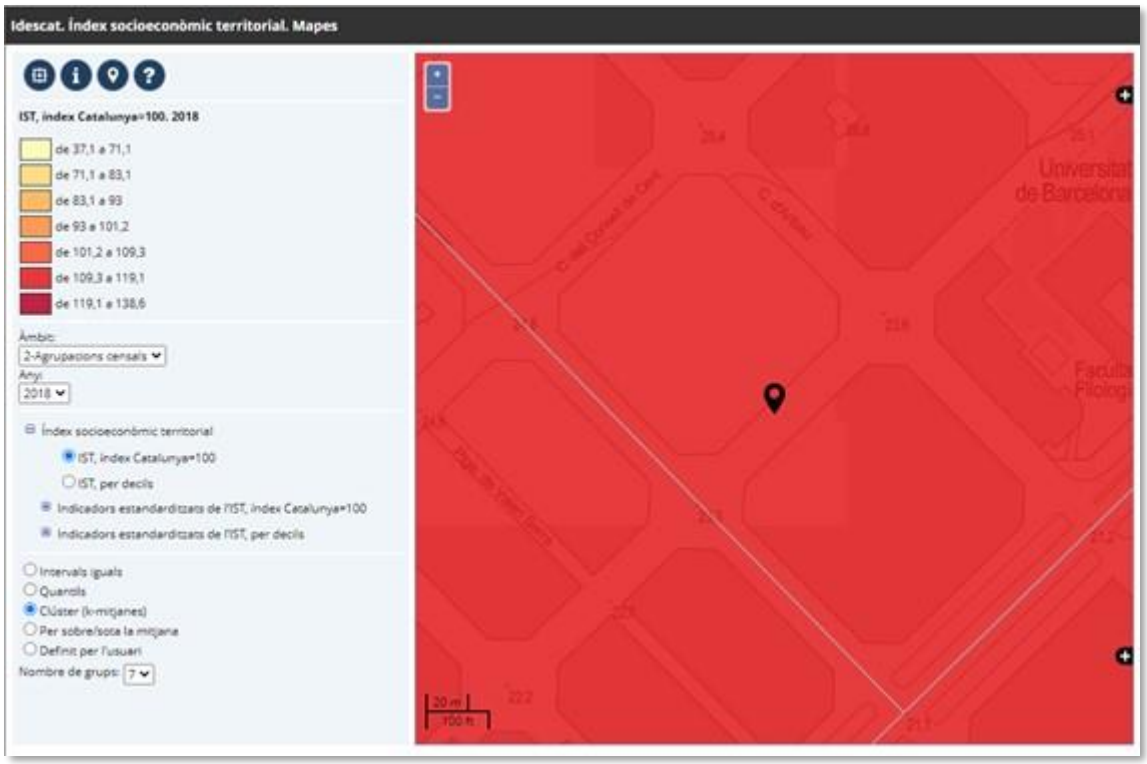

Al clicar en alguno de los valores atípicos detectados se produce una traslación y ampliación a esa zona geográfica:



Además, el usuario puede posicionarse en el mapa mediante la geocodificación de una dirección postal:



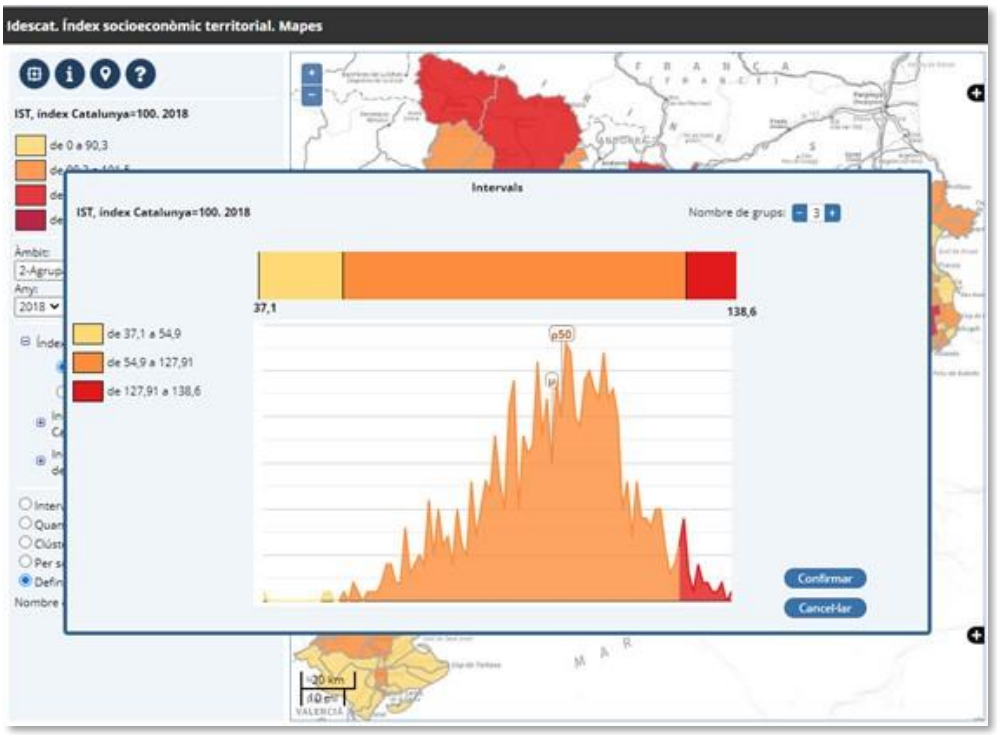

realizando la aplicación una traslación y un zoom al punto obtenido:



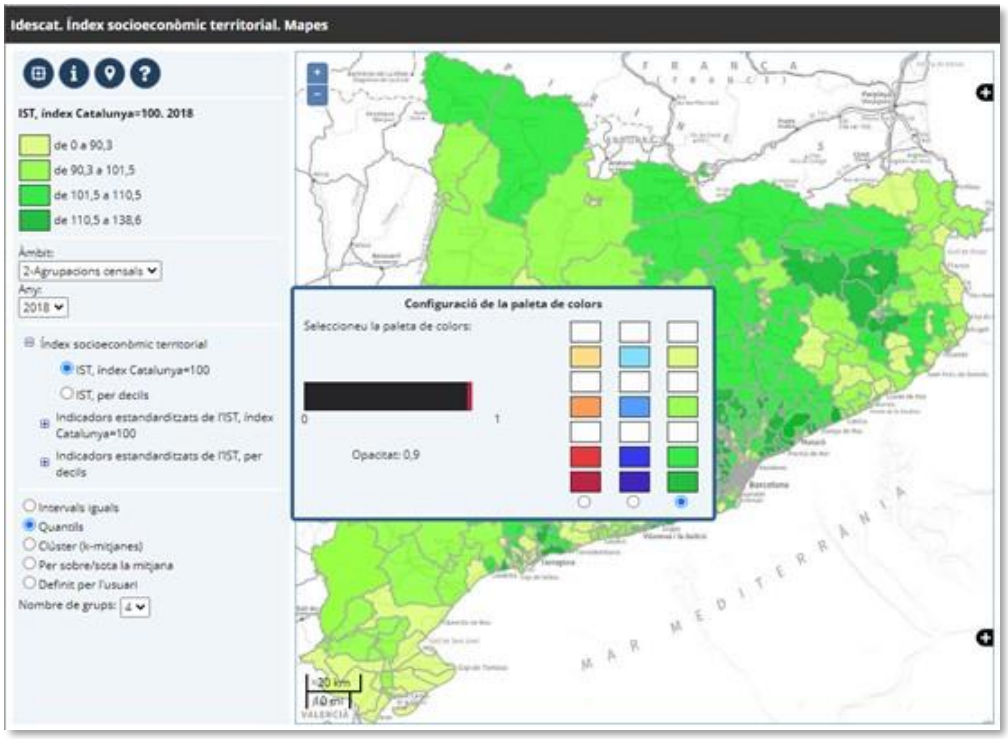

Este proceso de geocodificación se realiza centralizadamente en el servlet **SLDGenerato**r que dispone de los métodos remotos correspondientes para comunicarse con los clientes

y del API SOAP para la comunicación entre él y el servicio de geocodificación del Institut Cartogràfic i Geològic de Catalunya (ICGC).

En cuanto a la definición de intervalos existe la posibilidad de que el usuario los establezca según sus propios criterios (mostrando el histograma de la variable en cuestión):




Y, para finalizar, el usuario puede modificar la paleta de colores y la opacidad de la representación:



Todas estas características han sido implementadas mediante un conjunto de clases Java que tratan los eventos relativos a las acciones del usuario.

Existen, además, otro subconjunto de clases (como la clase **Histograma** o **ArgumentsInfoPunt**) cuya finalidad es el paso de argumentos o de valores devueltos en las llamadas asíncronas AJAX. Existen tanto en el servidor como en el cliente, unos como código Java y otros como código JavaScript, pero al nivel de abstracción del desarrollador son indistinguibles.

Una vez compilado el conjunto de clases por parte de Java, el compilador GWT convierte la parte cliente del código en un bloque de código JavaScript, optimizado para cada tipo de navegador. El compilador GWT detecta que clases Java debe convertir a JavaScript en función del package en donde residen.

Cualquier acción de modificación, depuración de errores, etc., se realiza en el entorno de Java que estemos utilizando (en nuestro caso Apache NetBeans [16]) sin tener en cuenta si el código va a ser Java o JavaScript, con lo que la seguridad y productividad aumentan considerablemente, comparándolo con el desarrollo típico (Java en el servidor y JavaScript en el cliente).

## 5. Conclusión

- El uso de GWT aumenta la fiabilidad, productividad y facilita el mantenimiento de aplicaciones Web mínimamente complejas

- El uso de software libre para la realización de aplicaciones como la que se ha descrito es factible y altamente recomendable

- Consideramos de utilidad para el usuario que para aplicaciones de esta naturaleza se obtenga, además de la información meramente cartográfica, información estadística básica como el histograma, los valores de tendencia central y de dispersión, la detección y localización de valores atípicos etc.

- Gracias a la definición de la meta base, este producto de software es totalmente general en relación con la generación de mapas de coropletas, utilizando el protocolo WMS, para variables estadísticas numéricas.

- Aunque la aplicación descrita implementa el conjunto completo de requerimientos expuestos en los objetivos y es lo suficientemente general, siempre cabe la posibilidad de ampliarlos. Esta decisión dependerá sobre todo de los costes de desarrollo implicados.

  De entre las posibles mejoras, razonables en cuanto a su costo, podríamos destacar:

  - Posibilidad de descarga de datos en formato GML o shape con selección de subconjuntos territoriales.

  - Implementación de algún recurso gráfico para señalar valores inferiores a un cierto umbral y que no deban ser difundidos (confidencialidad). A nivel de la meta base existen las estructuras necesarias para realizar esta acción, pero no a nivel de recursos gráficos, aunque su implementación es relativamente fácil.

  - Cálculo de índices de correlación espacial y localización de valores atípicos espaciales. Para ello sería necesario el cálculo previo de índices (por ejemplo, la I de Moran [17]), decidiendo que tipo de distancia se utilizará en el cálculo, su almacenamiento en la base de datos y el cálculo de ciertos diagramas (gráficos de dispersión de Moran) para la posible localización de valores atípicos espaciales [18]. Disponer de estos datos pre calculados tiene la ventaja de poder mostrar los resultados sólo para aquellas variables que tengan una fuerte correlación espacial. Esta mejora, contrariamente a lo que pueda parecer, tiene un coste mínimo ya que disponemos del código Java desarrollado ya hace algunos años.

## 6. **Referencias**


[1] http://geoserver.org/

[2] https://postgis.net/

[3] https://openlayers.org/

[4] http://tomcat.apache.org/

[5] https://www.ogc.org/standards/wms

[6] http://www.gwtproject.org/

[7] MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. **1**. University of California Press. pp. 281–297

[8] Statistical disclosure control on visualizing geocoded population data using a structure in quadtrees. Techniques and Technologies for Statistics (NTTS). European Commission.
https://ec.europa.eu/eurostat/cros/ntts2017programme/data/abstracts/abstract_286.htm

[9] https://www.cs.waikato.ac.nz/ml/index.html

[10] Le Hégaret, Philippe (2002). The W3C Document Object Model (DOM). World Wide Web Consortium.

[11] https://www.oracle.com/database/technologies/spatialandgraph.html

[12] https://www.ogc.org/standards/sfa

[13] https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/

[14] Guttman, A. (1984). R-Trees: A Dynamic Index Structure for Spatial Searching. Proceedings of the 1984 ACM SIGMOD international conference on Management of data – *SIGMOD '84*. p. 47

[15] http://www.gwtopenlayers.org/

[156] https://netbeans.apache.org/

[17] Moran, P. A. P. (1950). "Notes on Continuous Stochastic Phenomena". Biometrika 37 (1): 17–23.

[18] https://geodacenter.github.io/workbook/6a_local_auto/lab6a.html