Copy and Paste between channels

In order to achieve copy and paste between channels it is necessary to write data to the OS clipboard on copy/cut action and then read data back from the OS clipboard on paste action. The Idea is to serialize both graph and model data into an XML/JSON then deserialize it into the target canvas.
By using the OS clipboard it will be possible to copy/cut elements (nodes and connectors) from VsCode plugin tab and paste it into some other tab of the same editor type(BPMN/DMN) and in between the online-editors and the VsCode as well.

Writing to the Clipboard

The fastest way to write text to the OS clipboard is by using the new native API detailed in the link below.

https://developer.mozilla.org/en-US/docs/Web/API/Clipboard/writeText

The tricky part is that it won't work if the API is called through the "Document" object. In this context the browser will complain that the document is not focused. However if you call the API through the "Window" object it will work as expected.
Check the browser compatibility table. It will work for Chrome/Firefox/Safari.

E.g.

```
public native boolean copyToClipboard(final String content) /*-{
      $wnd.navigator.clipboard.writeText(content).then(function () {
         //Clipboard successfully set
         console.log("Text copied to clipboard");
      }, function (error) {
         //Clipboard write failed
         console.log("Copy to clipboard failed: " + error);
      });
 }-*/;
```

Note: The code above was written with JSNI for a quick test, however the same can be achieved with JsItnterop.

Reading from the the Clipboard

The fastest way to read text to the OS clipboard is by using the new native API detailed in the link below.

https://developer.mozilla.org/en-US/docs/Web/API/Clipboard/readText

Check the browser compatibility table. It will work for Chrome/Safari but Firefox. Firefox is not supporting this API yet and allows only browser extensions to read from the OS clipboard.

One option is to go with the new API and leave the Firefox out until it supports the new API.

One other option is to implement a workaround based on the old API (execCommand('paste')) and it should work in Chrome/Safari/Firefox.

In this approach the idea is to trigger and then intercept the ‘paste’ event and finally extract the clipboard data from within the event.
The downside is that we need to attach a DOMListener to some focusable element and then after getting the data remove it. It works well with a div element in Chrome, however it won’t with firefox. To get it working with firefox we will have to trick the browser by temporarily adding a TextArea into a div. Maybe make it very small and not visible to the user, nonetheless the TextArea must be still focusable. That being said, we attach the “paste” listener to the text area, focus the textArea and then we trigger the paste event.
E.g.

```
public native void pasteFromClipboard() /*-{
      //This works
      $doc.getElementById("myTextarea").addEventListener("paste", function (e) {
         e.preventDefault();
         var text = e.clipboardData.getData("text/plain");
         console.log("Text from clipboard event--:" + text);
      });

      $doc.execCommand('paste');
 }-*/;
```

GWT does not support it natively, however, it might be with elemental.
E.g.

```
 …
EventListener pasteListener = e -> handlePaste(e);
…
textArea.addEventListener("paste",pasteListener);
...
private void handlePaste(final elemental2.dom.Event event) {
      ClipboardEvent clipboardEvent = Js.uncheckedCast(event);
      String text = clipboardEvent.clipboardData.getData("text");
      DomGlobal.console.log("Text from elemental: " + text);
}
```

The problem with the elemental2 approach is that the cast of the GWT canvas div is not working directly, however, it should work in some other way.
This is not working for some reason:
HTMLDivElement panel = Js.uncheckedCast(getCanvasHandler().getAbstractCanvas().getView().getPanel().asWidget().getElement());

Serialization and Deserialization

GWT obfuscates the name of the object fields so it won't be possible to serialize and deserialize data without adding JsTypes to the objects we want to serialize.

Considering client code, the most feasible way to serialize and deserialize js objects is by using the native interface JSON.stringfy and JSON.parse.
E.g.

```
@JsType(isNative = true, namespace = "com.mypackage")
public class JSON {

    public static native String stringify(Object obj);

    public static native Object parse(String obj);
}
```

It will work with custom types as below:

```
@JsType(isNative = false, namespace = "com.mypackage")
public class Customer {

    public String name;
    public Object someObj;

    public Customer() {}
}
```

Deserialization works fine when we know the types beforehand and we can do the cast.
The biggest issue here is that once deserialized, the original object class name is lost and if you check the type it will be "com.google.gwt.core.client.JavaScriptObject" and there is extensive use of "[object].getClass()" to solve types in stunner in order to make the code generic.
E.g.

```
...
DomGlobal.console.log("customer type before: " + customer.getClass().toString());
String objJSON = JSON.stringify(customer);
DomGlobal.console.log("-> " + objJSON);
```

Object deserializedObj = JSON.parse(objJSON);
DomGlobal.console.log("Object type: " + deserializedObj.getClass().toString()); // no type

Customer customerDeserialized = Js.uncheckedCast(deserializedObj); // we know the type
…

There aren't many sources of information about deserialization with JSON native API and most of the examples use js “Object” type instead of custom types.
E.g.

```
@JsType(isNative = true, namespace = JsPackage.GLOBAL, name = "Object")
public class Customer {

    public String name;
    public Object someObj;

    public Customer() {}
}
```

Blockers

1. We must have all domains (BPMN/DMN) as native js objects (exported to jsinterop), but they're actually not yet and this depends on the marshallers work (BAPL-1658).
2. Even once having all types exported to js - We have to solve the JSON.parse, which returns an object which class is "com.google.gwt.core.client.JavaScriptObject", but not the classname of he bean we expect.
3. Decide which approach to follow in regards with the paste APIs. Go with the new API (not supported at the moment by Firefox) or go with the old deprecated API and trick the Firefox catching the paste event as mentioned earlier in this document.