Lienzo testing

0 - Executive summary

In the light of increasing number of features relying on the use of Lienzo library [1] for their implementation, QE has performed some initial research on Canvas testability.
This document summarizes results of our research. It turns out that unless we follow best practices and implement some code to support testing we'll have very hard time implementing any high-level functional tests. There are several findings as well as a word of warning we would like to share.

1 - Canvas accessibility & best practices

We want to emphasize that canvas shouldn't be used to render complex interactive controls like text editing fields and selection dropdowns. Such approach would be testing unfriendly and error-prone. The good example is the Guided decision tree editor, where the canvas interaction is restricted to Drag & Drop of nodes, node selection and clickable context buttons (edit, delete, collapse/expand). Guided decision table editor is probably on the right path, too.

"The <canvas> element on its own is just a bitmap and does not provide information about any drawn objects. Canvas content is not exposed to accessibility tools like semantic HTML is. In general, you should avoid using canvas in an accessible website or app." [2]

"Authors should avoid implementing text editing controls using the canvas element." [3]

There are 2 accessibility hooks on the Canvas API level, both of which are out of our scope:

- Fallback content - DOM tree under the <canvas> element that can be displayed to user using a browser that doesn't support canvas. But we don't have such users (from product point of view).
- HitRegions - an arbitrary area that can be mapped to any shape drawn on the canvas, which can route mouse/keyboard events to an element under the current <canvas>. Sounds like a powerful tool which can be used to mimic MVC (canvas is the View, elements receiving the events are the Controller+Model). But this is an experimental Canvas feature, not enabled by default in browsers ("behind a flag") [4] and **not covered by Lienzo API**. Take a look at this demo [5] if you want to see HitRegions in action (you

must set canvas.hitregions.enabled=true in Firefox’s about:config or ExperimentalCanvasFeatures=true in Chrome).

2 - QE & Selenium Testing

QE will aim to create basic functional test coverage of Lienzo-based editors using selenium. Unfortunately, **the structure of HTML Canvas element is off limits to selenium**. I.e. selenium can “see” the canvas element itself, but the structure of the graphics inside the canvas has no extractable programmatic representation whose contents could be checked automatically - it’s similar to an image.
To test it as it is now used in the workbench, we would have to compute (x,y) coordinates for DnD and click interactions, which (as our experience shows) is very brittle, error prone and in some cases so hard as to be impractical to model in tests (e.g. Guided decision tree nodes are automatically reorganized after node is added / removed, which changes the coordinates of all the nodes in canvas).
For that reason it is desirable, that most of user interactions happen outside of canvas, e.g. in wizards, simple edit dialogs, confirmation modals etc. that are part of DOM outside of Canvas.

There is a way that engineering could help us to eliminate computing of coordinates of elements that are already in the canvas, which is to implement **JSON serialization support**. This would mean to follow the recipe for extending Lienzo [6] and implement serialization support for custom classes extending com.ait.lienzo.client.core.shape.Node (e.g. classes in Wires and specific editors like DTree, DTable and BPMN). The last step would be a mechanism allowing us to obtain the serialized model [7] on demand (via some JavaScript function that would need to be added for the sake of testing).

The main benefit of having JSON model of the nodes rendered in canvas would be the **programmatic access to the canvas content** - e.g. we could count how many nodes there are, what are the links between them and other properties, that will be otherwise programmatically inaccessible.

Other options of canvas content verification are usage of the Source tab of editors (very indirect) or “blind” canvas interaction (i.e. to click a coordinates where a node’s edit button should be and if a dialog appears, we can be sure the node was present). But this is sure to lead to creation of very brittle tests and to some serious test maintenance overhead in the future.

3 - Unit Testing

Good code **coverage in community is critical** to compensate for the lack or more comprehensive Selenium testing. Most of both Eng and QE effort should focus there. Developers should enable code testability by creating some basic tests. QE can then improve coverage, add bugfix (regression) tests. If that's the case, selenium testing wouldn't be that critical and could be minimized to avoid maintenance overhead.

4 - Summary & Recommendations

- DON'T use canvas for stuff that's can be done through plain GWT / HTML.
- DO Try implement maximum functionality outside-of-canvas.
- If Selenium tests of canvas functionality is to be possible, we'll need engineering to implement some JSON serialization support & javascript hooks to obtain information about the structure of nodes in the canvas.
- Community unit / integration test coverage will be more important for components using Lienzo than anywhere else.

Q & A

Q1: Would engineering be willing to accommodate selenium testing and implement canvas node serialization support + some javascript hook to enable obtaining JSON model of the nodes drawn to canvas?

[1] Lienzo (spanish for "canvas") is a structured graphics toolkit for GWT. It allows you to build scalable vector graphics applications (in Java) that will run in any browser or device that supports the HTML5 Canvas API. At the moment it's used in: Guided Decision Tree editor, Guided decision table editor, some charts in Dashbuilder and most importantly the new Process Designer is going to be based on it.
[2] Hit regions and accessibility
[3] Canvas - best practices
[4] Hit regions - browser compatibility
[5] Hit regions - example
[6] Extending lienzo - serialization
[7] https://www.lienzo-core.com/lienzo-ks/#WELCOME - click JSON tab