User Interface - Bug #3597

fix direct manipulation problems related to MOVABLE, SELECTED, SELECTABLE and misc widget types

05/30/2018 08:51 AM - Constantin Asofiei

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	50%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to User Interface - Bug #4671: Widget drag doesn't work in web	**New**	**06/13/2020**

History

#1 - 05/30/2018 08:53 AM - Constantin Asofiei

There are a couple of problems in FWD's implementation of direct manipulation. A few will be fixed in #3567, but what remains is basically combination(s) of:

1. widgets with MOVABLE = true
2. widget type i.e. TEXT, FILL-IN, BUTTON, etc
3. SELECTABLE = FALSE and SELECTED attribute
4. when there are multiple MOVABLE widgets, you can't get rid of the previously moved widget: once you select another widget to move, the previous one will be included in the 'movable' list and will be moved, too. 4GL doesn't do this.

#2 - 05/30/2018 09:42 AM - Constantin Asofiei

The movable widgets is not working in Web - see #3567-162 for the recreate.

#3 - 07/12/2018 11:18 AM - Eric Faulhaber

- Assignee set to Eugenie Lyzenko

#4 - 07/12/2018 12:43 PM - Constantin Asofiei

At some point there was some discussion about the fact that the move/resize is not 'live' in 4GL (i.e. entire widget is not resized/moved on mouse move); instead 4GL uses a boundary rectangle, and when dragging is finished, the actual resize/move is performed. Currently, this boundary rectangle is being drawn via the client-side - in Web, this is almost unusable, as with each mouse move the UI is redrawn (which can be even worse on slow network connections). We need to move this at the driver-side (i.e. draw the boundary rectangle there, and let the driver resize/move it, with no trips into the FWD client-side code) - FWD client-side will do the resize/move only once the operation at the driver level has finished (when mouse dragging has finished?).

Eugenie, do you think this is something which can be decoupled, so that the Swing and Web each have their own implementation?

#5 - 07/12/2018 04:44 PM - Eugenie Lyzenko

Constantin Asofiei wrote:

> At some point there was some discussion about the fact that the move/resize is not 'live' in 4GL (i.e. entire widget is not resized/moved on mouse move); instead 4GL uses a boundary rectangle, and when dragging is finished, the actual resize/move is performed. Currently, this boundary rectangle is being drawn via the client-side - in Web, this is almost unusable, as with each mouse move the UI is redrawn (which can be even worse on slow network connections). We need to move this at the driver-side (i.e. draw the boundary rectangle there, and let the driver resize/move it, with no trips into the FWD client-side code) - FWD client-side will do the resize/move only once the operation at the driver level has finished (when mouse dragging has finished?).
>
> Eugenie, do you think this is something which can be decoupled, so that the Swing and Web each have their own implementation?

Constantin, I do not forget your question, just thinking. This is a complex point to make fast decisions or conclusions. Please wait I need to complete some work for border related issue.

#6 - 07/16/2018 09:43 PM - Eugenie Lyzenko

Constantin Asofiei wrote:

> At some point there was some discussion about the fact that the move/resize is not 'live' in 4GL (i.e. entire widget is not resized/moved on mouse move); instead 4GL uses a boundary rectangle, and when dragging is finished, the actual resize/move is performed. Currently, this boundary rectangle is being drawn via the client-side - in Web, this is almost unusable, as with each mouse move the UI is redrawn (which can be even worse on slow network connections). We need to move this at the driver-side (i.e. draw the boundary rectangle there, and let the driver resize/move it, with no trips into the FWD client-side code) - FWD client-side will do the resize/move only once the operation at the driver level has finished (when mouse dragging has finished?).
>
> Eugenie, do you think this is something which can be decoupled, so that the Swing and Web each have their own implementation?

Let's clarify some points here.

If my understanding is correct the problematic areas are the drawing the dotted widgets boundary frames in Web client on selected widgets move/resize, correct?

As the background info, the Swing and Web clients are implemented differently starting with GuiDriver calls. For example the main worker - drawSelectedArea() works completely on the driver level in Swing client (SwingGuiDriver does the real painting). For the Web driver the situation is more complex, the painting is performing via websocket calls. Here I can agree, the network performance is a point. But anyway the real widget moving is performing when the action completes (drag is finished).

So what are you asking for possibility to rework? Consider the following scenario:
1. Web client receives mouse drag start for selectable widgets. This sends start command to JS (JavaScript) engine via web socket.
2. Until mouse drag finishes, all required paintings inside browse canvas is performing by JS system, neither client nor websocket are involving in this process.
3. When mouse drag finishes the JS send message to client via websocket about the message finished.
4. Client initiates respective direct manipulation event.

It this scenario what you expecting to have after rework, correct? And the issue is only to paint dotted widgets boundary rectangle, right?

From general perspective we need to rework Web client part only, no Swing painting code needs to be changed. For Web client decoupling - there are several issues I see here.
1. We need to suspend client code execution on the time of delegating direct manipulation work to JS, right or not?
2. Is it possible to find required functionality inside pure JS to implement all boundary rectangles painting?

#7 - 07/18/2018 02:47 AM - Constantin Asofiei

Eugenie Lyzenko wrote:

> If my understanding is correct the problematic areas are the drawing the dotted widgets boundary frames in Web client on selected widgets move/resize, correct?

Yes.

> As the background info, the Swing and Web clients are implemented differently starting with GuiDriver calls. For example the main worker - drawSelectedArea() works completely on the driver level in Swing client (SwingGuiDriver does the real painting).

I'm not sure I understand - there is no drawSelectedArea() method - you mean MouseDirectManipulation.drawSelectionBox()? As a side note, this drawing code looks like is being invoked from within the event processing loop; generally we don't want drawing to be performed here, we rely on repaint() to mark widgets which need to be redrawn. I understand this is a special case, where no widgets are being used, so for Swing leave it like this if it works properly.

> It this scenario what you expecting to have after rework, correct? And the issue is only to paint dotted widgets boundary rectangle, right?

Yes, exactly - the dotted boundary painting must be done fully on JS side, without intermediate drawing API calls from the client-side via web-sockets.

> From general perspective we need to rework Web client part only, no Swing painting code needs to be changed. For Web client decoupling - there are several issues I see here.
> 1. We need to suspend client code execution on the time of delegating direct manipulation work to JS, right or not?

The JS side needs to be in a state where the mouse events are not being sent via the web socket until the mouse dragging has finished. The client-side, even if it is 'parked' in the event-processing loop (i.e. wait-for) at the start of the dragging, it should wait for mouse dragging to finish before resuming.

> 2. Is it possible to find required functionality inside pure JS to implement all boundary rectangles painting?

The drawing APIs are in p2j.canvas_renderer.js; the boundaries can be sent from client-side.

#8 - 07/18/2018 08:48 AM - Eugenie Lyzenko

Constantin Asofiei wrote:

> I'm not sure I understand - there is no drawSelectedArea() method - you mean MouseDirectManipulation.drawSelectionBox()?

Yes, exactly, sorry for confusion. MouseDirectManipulation.drawSelectionBox() is what I did mean.

#9 - 07/18/2018 08:50 AM - Constantin Asofiei

Another implementation note: the JS side will need to save the current window canvas and the dotted rectangle will be drawn over this. Otherwise you can't re-draw the widgets which were affected by the previous dotted rectangle (before the move/resize).

#10 - 11/13/2018 09:11 PM - Eugenie Lyzenko

- Status changed from New to WIP

Resume working on this branch.

#11 - 11/14/2018 10:41 PM - Eugenie Lyzenko

Making a review and check for the current status of the direct manipulation functionality to create . The testcases I'm using: drag_n_drop/dnd_test*.p. And I do not see the issues or TODO work other than changing selection boxes painting implementation for Web client.

Constantin, if you see any other issues please provide the scenario or event better the testcase to reproduce.

Creating the implementation plan for JS Web client specific changes.

#12 - 11/15/2018 06:31 PM - Eugenie Lyzenko

Questions to clarify:

Is the p2j.screen.js the main module to capture and handle JS mouse events? What is the purpose of the p2j.mouse.js? We have two mouse handling JS modules? Or there is a chain when mouse event from JS is handling first in p2j.mouse.js and then may be in p2j.screen.js and then if not consumed - be passed to the Java client side?

I need to have complete clear picture for Web driver mouse handling to design new direct manipulation Web driver specific that can be more effective than currently used unified(for Web and Swing clients) approach.

#13 - 11/15/2018 06:35 PM - Constantin Asofiei

Question: when moving/resizing a widget, is it possible to have its selection rectangle drawn behind a widget, while this operation is processing? Because if this is possible, doing this on JS side is not a solution. The JS-side solution assumes the selection rectangle (which is being moved/resized) is always above all widgets, and can never exist (and draw) behind other widgets.

#14 - 11/15/2018 06:43 PM - Eugenie Lyzenko

Constantin Asofiei wrote:

> Question: when moving/resizing a widget, is it possible to have its selection rectangle drawn behind a widget, while this operation is processing? Because if this is possible, doing this on JS side is not a solution. The JS-side solution assumes the selection rectangle (which is being moved/resized) is always above all widgets, and can never exist (and draw) behind other widgets.

I'm not sure I've completely got your question. In JS the selection boxes are always painting in separate canvas which is on the top of all other(having widgets). So in this transparent canvas only selection rectangles exist, no widgets at all.

#15 - 11/15/2018 06:46 PM - Constantin Asofiei

Eugenie Lyzenko wrote:

> I'm not sure I've completely got your question. In JS the selection boxes are always painting in separate canvas which is on the top of all other(having widgets). So in this transparent canvas only selection rectangles exist, no widgets at all.

I mean in 4GL. If 4GL allows the selection box to draw behind another 4GL widget, then we have a problem with this approach, as we will not be fully compatible.

#16 - 11/15/2018 06:52 PM - Constantin Asofiei

Eugenie Lyzenko wrote:

> Is the p2j.screen.js the main module to capture and handle JS mouse events?

Yes, this is responsible for capturing and dispatching the mouse event to the Java side; but these are simple events (like clicks), and not high-level actions (like window move).

> What is the purpose of the p2j.mouse.js? We have two mouse handling JS modules?

This provides implementations of high-level mouse actions (like moving a window) - which do not require trips to the Java side for each and every mouse action, only when the action has completed.

> Or there is a chain when mouse event from JS is handling first in p2j.mouse.js and then may be in p2j.screen.js and then if not consumed - be passed to the Java client side?

Exactly; if p2j.mouse.js gets the event first, it will process (like a window move), and at the end will notify the Java side with i.e. the window's new dimension.

You need something similar - you get in a state where you are dragging the mouse (and moving/resizing the selection box), and once this is done, inform the Java side of the new coordinates/dimensions; p2j.screen.js will not be involved in mouse notifications, just the high-level action in p2j.mouse.js.

#17 - 11/15/2018 07:04 PM - Eugenie Lyzenko

Constantin Asofiei wrote:

> Eugenie Lyzenko wrote:
>
> > I'm not sure I've completely got your question. In JS the selection boxes are always painting in separate canvas which is on the top of all other(having widgets). So in this transparent canvas only selection rectangles exist, no widgets at all.
>
> I mean in 4GL. If 4GL allows the selection box to draw behind another 4GL widget, then we have a problem with this approach, as we will not be fully compatible.

In 4GL the selection box is painting over the current content(XOR-ed). Like in native DnD operation. The dragged object frame is always on the top of everything for user to see what is happening.

#18 - 11/16/2018 02:00 PM - Eugenie Lyzenko

After some investigation and design analysis my refactoring plan is(the implementation can be different, this is the base idea):

1. The only thing that need to be different is MouseDirectManupulation.mouseDragged() handler. Here the code that do actual selected rectangles. This is the place where we decide the drag has been started. We need(and able) to have driver dependent implementation here.
2. For Swing client the MouseDirectManupulation will use the current code that is known as well working.
3. For Web client we will have class that is based on MouseDirectManupulation and implement specific mouseDragged() handler.
4. The only thing for Web client the Java code will do for mouseDragged() is to notify JS the direct manipulation started. This is probably the moment when we can pass the selection rectangles set from Java to JS. Then further control will be passed completely to JS driver until the JS detects and send end of the process message back to Java.
5. After receiving end of the process message the direct manipulation will be common again to process actual widgets move/resize.

#19 - 11/20/2018 10:12 PM - Eugenie Lyzenko

- % Done changed from 0 to 50

Task branch 3782a has been updated for review to revision 11321.

This update introduces the new approach for Web client direct manipulation rework. There a significant work to do here, some features are in

development(snap to grid, row resize adjustments). Some features still do not work as expected and require more debugging. But the base idea is here.

The concept is:
1. When the direct manipulation is about to be started(mouse press with valid result) the Java side sends the event to JS with all required info to perform selection boxes drawing purely in JS code.
2. When mouse release is happening - everything returns to the regular way.
3. Some operations will require one time drag call from JS to Java. The example is box selection when we have to send the starting message only inside Java drag handler. In this case appropriate pending one time call is included.

There are the issues to debug with widget resize related to mouse event processing. Hoping to find a solution soon and prepare completely working release candidate in a next day or two. This time is required to also add missing features like snap to grid support.

#20 - 11/21/2018 08:45 PM - Constantin Asofiei

Eugenie, please experiment what happens if there are child frames and the selected elements (to be moved) are not from the same frame.

#21 - 11/21/2018 09:51 PM - Eugenie Lyzenko

Constantin Asofiei wrote:

> Eugenie, please experiment what happens if there are child frames and the selected elements (to be moved) are not from the same frame.

OK. Actually there are serious issues I have detected in Web client case. The reason is the event synchronizing for mouse events happening in JS and Java sides. Moreover the JS uses setTimeout() async call to simulate drag event on mouse move, complicating the situation even more. I need a bit more time than I've expected to find out the solution.

#22 - 11/22/2018 07:09 PM - Eugenie Lyzenko

Task branch 3782a has been updated for review to revisions 11323, 11324.

This is some stable pause point to be able to safely postpone further debug and modifications.

Also fixed small javadoc issue in Frame.java and cancelDrag() implementation in p2j.screen.js.

Unfortunately the implementation is not complete. Actually the only thing that works is selection box drawing and respective widget selections. Than more I debug into this implementation than more troubles I see here. So after gathering the info I need a time to think. But there are the good news too. The main issue I think is concurrency issue for widgets that can simultaneously request JS for box selection mode. In FWD we register direct manipulation handler for every widget(including frame). If we start operation(press) in one widget and end with another(release) - the issue is starting because every widget currently has it's own set of direct manipulation data. But per session only one operation is possible. So far we need to add some static code to control the process from the single point.

I have made the first steps but it will take a time to prepare full stable release. So due to time constraints I'm making a pause for now to shift to more priority task to solve. I'll return to the task when get enough time.

#23 - 11/13/2019 12:07 PM - Greg Shah

- Assignee deleted (Eugenie Lyzenko)

The changes from 3782a are in trunk (12 months ago). The rest of the work remains open.

#24 - 06/13/2020 03:47 PM - Greg Shah

- Related to Bug #4671: Widget drag doesn't work in web added