User Interface - Feature #3246

reduce the amount of data being sent to the client-side when an UI attribute is being changed

02/09/2017 01:55 PM - Constantin Asofiei

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to User Interface - Feature #3394: generic drawing improvements: repa...	**Closed**		
Related to User Interface - Bug #2887: web client flashing/drawing artifacts	**Closed**		
Related to User Interface - Bug #2766: implement a text metrics server process	**New**		
Related to User Interface - Support #2672: GUI web client performance improve...	**New**		
Related to User Interface - Bug #2613: improve performance of ScreenBitmap.ca...	**New**		
Related to Runtime Infrastructure - Feature #2973: improve ContextLocal perfo...	**Closed**		
Related to User Interface - Feature #1791: implement dynamic UI support	**Closed**	**02/21/2013**	**11/08/2013**

History

#1 - 02/09/2017 02:08 PM - Constantin Asofiei

- Start date deleted (02/09/2017)

Whenever a widget attribute setter is being called on server-side, FWD relies on the pushScreenDefinition() to notify the client-side about the change, which will push to the client side the entire frame definition, even if only one attribute has changed (and maybe the attribute hasn't changed at all); when client and server are on different machine, this can cause problems depending on the network setup. For ADM frames, the UI tree may be very complex (many nested frames, widgets, etc), and also these setters may be called a lot of times (i.e. when widgets are created in a loop).

The approach when a attribute setter is being called needs to be improved, and push to the client-side only that widget's id, the attribute (the id of the field, as we do in ClientConfigManager?) and the value of the attribute. Attempts at using AspectJ on server-side when the ConfigManager was introduced led to performance degradation - mainly, because attribute setters are being called during static frame setup, too, and AspectJ I think introduced bytecode overhead which affected this case; but maybe it was just an incorrect approach when I tested it, and AspectJ might still work if is 'tweaked' better.

Another case which needs to be accounted for is when a dynamic widget is not yet attached to a frame: in this case, attribute changes need to remain on server-side, but when the widget is attached to the frame, we should avoid pushing the entire frame definition to the client, and just this widget definition.

#2 - 01/03/2018 02:03 PM - Greg Shah

- Related to Feature #3394: generic drawing improvements: repaint and other misc code added

#3 - 01/03/2018 02:05 PM - Greg Shah

- Related to Bug #2887: web client flashing/drawing artifacts added

#4 - 01/03/2018 02:05 PM - Greg Shah

- Related to Bug #2766: implement a text metrics server process added

#5 - 01/03/2018 02:05 PM - Greg Shah

- Related to Support #2672: GUI web client performance improvements added

#6 - 01/03/2018 02:07 PM - Greg Shah

- Related to Bug #2613: improve performance of ScreenBitmap.canDraw (and of its callers) added

#7 - 01/03/2018 02:11 PM - Greg Shah

- Related to Feature #2973: improve ContextLocal performance added

#8 - 01/03/2018 02:11 PM - Greg Shah

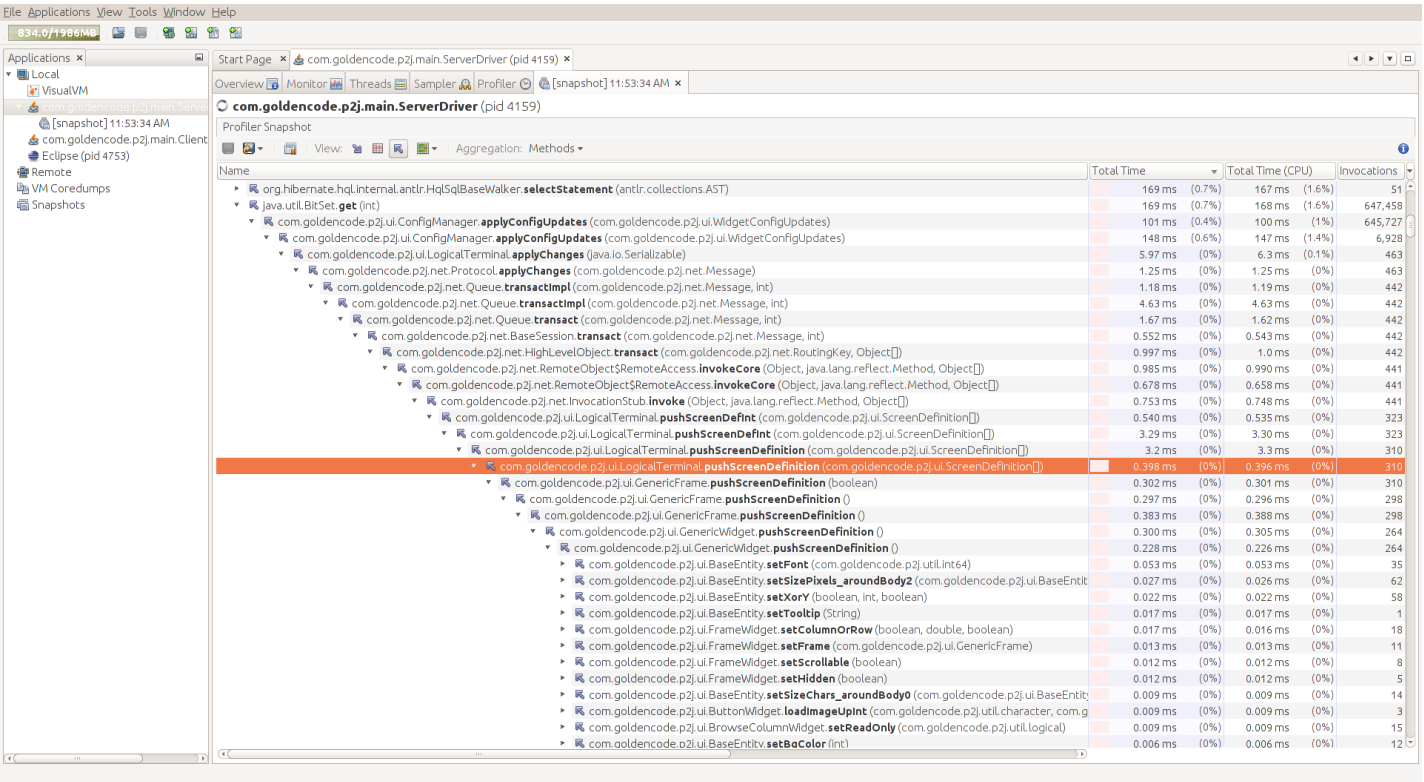
- Related to Feature #1791: implement dynamic UI support added

#9 - 01/03/2018 03:36 PM - Greg Shah

- Subject changed from reduce the amount of data being sent to the client-side when an UI attribute is beeing changed to reduce the amount of data being sent to the client-side when an UI attribute is being changed

- File profiling_bitset_get_method_usage_20180103.png added

The following screenshot was taken by ECF during his work on profiling the initial load of a complex ADM screen:



The full profiler snapshot is a 22MB file that doesn't compress well. It also has customer-specific details so it is not suitable for uploading here. It is available for review if needed.

This content of this screenshot is being posted here to serve as a basis for some discussions. This is an outgrowth of a discussion I had with Eric when he noted how often the BitSet.get() is being called on the server side (over 600,000 times just to load one screen). This is certainly something we can optimize. But as I dug into that use case, several things stood out and I want to discuss them here.

We have known for a long time that pushScreenDefinition() is very inefficient. As noted above, the use of AspectJ to optimize caused problems and so we evolved into two different approaches for synchronizing changes (i.e. server to client is different from client to server). The extreme inefficiency of pushScreenDefinition() is made worse by the fact that it is probably called when it is not needed AND it is synchronous. This trip down to the client can in turn cause a trip down to the JS side when running in the web client, which makes things worse. Constantin has tried to remove pushScreenDefinition() calls in many places, but if I remember correctly there were enough problems created that only a few places were able to be removed.

Of course, there is a time when the entire screen definition (graph of widget configs) does need to be pushed to the client. But this should be avoided for the case of individual attribute assignments.

Constantin:

- Is my recollection correct?
- Is it also correct that if we can make a big reduction in the cost of these state synchronization calls, that the performance will be noticeably improved?

It may be time to resolve this.

Instead of looking at AspectJ, perhaps we can come up with an alternate approach. What about a generic attribute setter that just transfers the new field value (if changed) to the client?

It seems like we can do this with the knowledge we already have in WidgetConfigDef. We can easily get back to the proper setter using the fid and the WidgetConfig class. Then we can call down to the client side, call the setter and return.

It might look something like LogicalTerminal.pushWidgetAttr(Class<T extends WidgetConfig> cls, int wid, String fname, Object value).

Instead of calling pushScreenDefinition(), attribute setter code on the server side would look something like this:

```
   public void setMovable(boolean movable)
   {
      if (config.movable == movable)
      {
         return;
      }

      config.movable = movable;
      LogicalTerminal.pushWidgetAttr(cfgCls, getId(), "movable", moveable);
   }
```

cfgCls would be cached as a member to avoid the (small but non-zero) cost of config.getClass().

LogicalTerminal.pushWidgetAttr() could use WidgetConfigDef to lookup the fid that matches the movable field. It seems this would be fast and would then be able to call down to the client which would have enough knowledge to set that value (and only that value).

Does anyone see a flaw in this logic?

Some other thoughts:

- In regard to the specific excessive use of BitSet.get(), I was surprised that pushScreenDefinition() would cause so much state synchronization to be pushed back up to the server. I can see some small amount of change driven by side effects on the client side, but this volume of changes seems to indicate that we are pushing way too much back over to the server. Perhaps we should be disabling dirty checking during the push?
- We can certainly also eliminate the use of BitSet, which is much more costly than direct bit manipulation, especially when called millions of times. I would have assumed that our client to server updates are "sparse". In other words, the client changes would only be for a small fraction of the total fields in any given widget config. If that is correct, then the WidgetConfigUpdates should probably just package up an array of container objects that simple hold the fid and changed value. Processing a small list like that might be less work than handling the BitSet.

#10 - 01/03/2018 04:26 PM - Hynek Cihlar

Greg Shah wrote:

> Instead of looking at AspectJ, perhaps we can come up with an alternate approach. What about a generic attribute setter that just transfers the new field value (if changed) to the client?

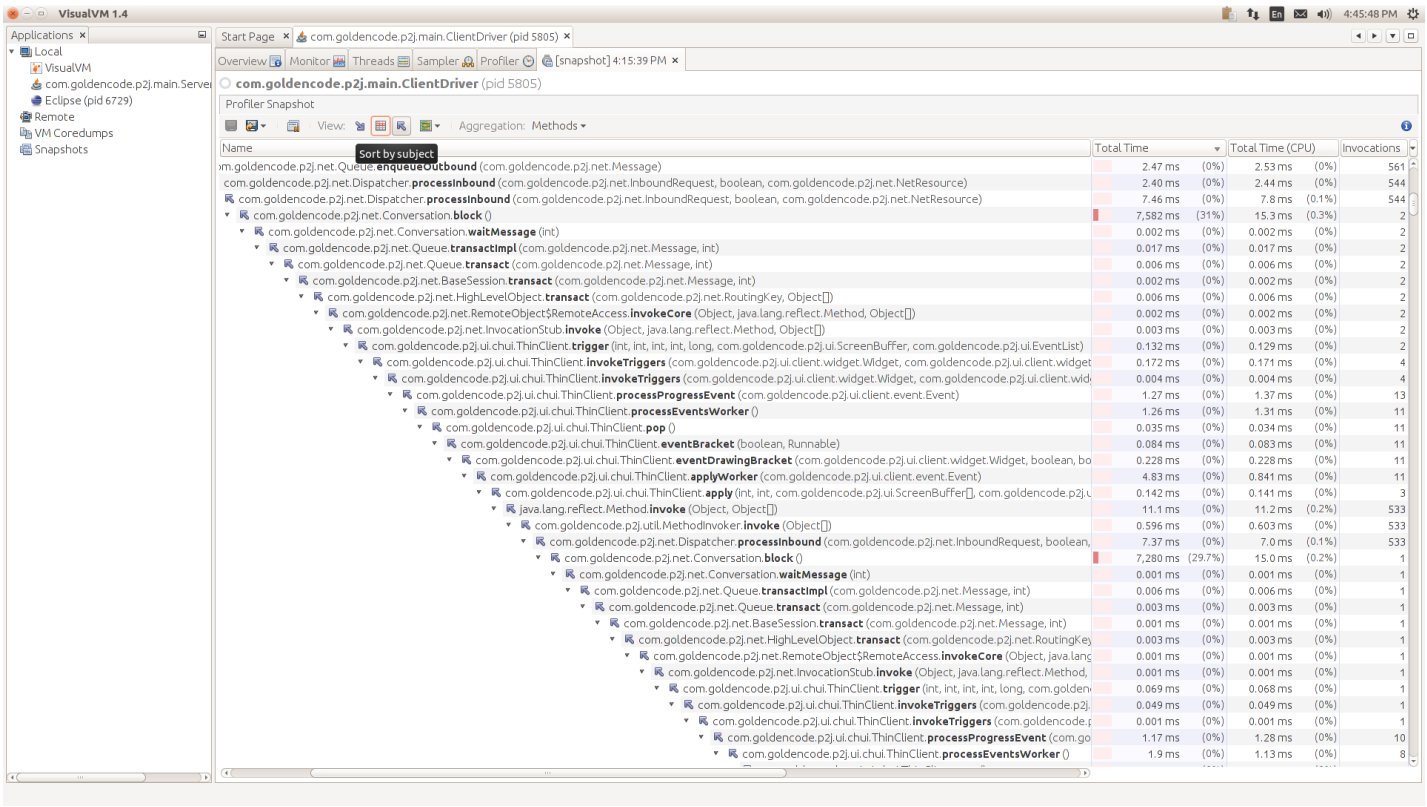
Instead of changing and hard-code every single setter I would consider to automate this and implement the common logic of the setter in AspectJ (i.e. the check for dirty and the remote call). This would be an iterative approach and so if the overhead of the AspectJ envelope was to high it could be replaced with something else.

I would avoid using Java reflection when setting the "pushed" config values. Java reflection, even when the reflection objects are cached, will introduce new overhead by itself. There are handy libraries built on ASM that provide "reflection" based on byte-code generated classes. One such lib is ReflectASM. Another advantage is that the fname could be replaced with a simple index (https://github.com/EsotericSoftware/reflectasm#avoiding-name-lookup).

#11 - 01/03/2018 04:55 PM - Eric Faulhaber

- File profiling_bitset_get_method_usage_client_20180103.png added

FYI, here's what the same use case looks like on the client, starting at BitSet.clear <- BitSet.set <- ConfigManager.getConfigUpdates <- ...



#12 - 01/03/2018 05:09 PM - Constantin Asofiei

Greg Shah wrote:

> Constantin:
>
> - Is my recollection correct?

Almost; this is always problematic when building an UI with dynamic widgets - see the calendar. In this case, there are lots of attribute assignments (especially location, size, sensitive, visible and/or hidden), which expose this performance issue easily, as is done in a tight loop. Beside the calendar, ADM still relies in certain cases on the same small sub-set of attributes (location, size, visibility and sensitivity) to layout/manipulate the UI. The problem we have here is that i.e. HIDDEN, VISIBLE and SENSITIVE are not simple setters - they affect the UI, in more complex ways (see the GenericWidget setters for these). But we may need to just move the behaviour directly on client-side... This was the show-stopper for me when I tried last to solve this (i.e. it stopped me from finding a quick way to fix it).

> - Is it also correct that if we can make a big reduction in the cost of these state synchronization calls, that the performance will be noticeably improved?

I expect so; there are two improvements here:

- network traffic: moving the entire frame tree for a simple attribute change from the server to the client is not optimal
- client-side code: the reconstruction (and processing) of each widget is expensive

But it all depends on how the ADM code is built - we may see improvements in some screens which relies heavily on widget attributes to layout the screen, and not so much in others (where the UI is static). OTOH, as we will not be touching each and every widget in the frame (during reconstruction), I expect to have a reduction of repaint calls in certain cases.

As a side note: we are not tracking changed fields, but assigned fields... as I recall it was pretty expensive to interrogate via reflection the previous field value (to send to server side only changed fields). So when ConfigManager.getConfigUpdates is reached, it will have to process all the config fields which were set (and not necessarily changed) by the client-side.

> - We can certainly also eliminate the use of BitSet, which is much more costly than direct bit manipulation, especially when called millions of times. I would have assumed that our client to server updates are "sparse". In other words, the client changes would only be for a small fraction of the total fields in any given widget config. If that is correct, then the WidgetConfigUpdates should probably just package up an array of container objects that simple hold the fid and changed value. Processing a small list like that might be less work than handling the BitSet.

I think you are correct here: this can be improved and eliminate the need for BitSet. This change can be made separately from the pushScreenDefinition improvements.

#13 - 01/04/2018 12:21 PM - Greg Shah

> I would consider to automate this and implement the common logic of the setter in AspectJ (i.e. the check for dirty and the remote call). This would be an iterative approach and so if the overhead of the AspectJ envelope was to high it could be replaced with something else.

Are you suggesting that we remove the server-side attribute setters or that we annotate them?

The setters themselves have a degree of variability, including some non-standard error checking/error generation, which would not fit well into a generic setter.

I wasn't suggesting replacement of the existing setter logic, just replacement of the pushScreenDefinition() call with something much faster/more efficient.

> I would avoid using Java reflection when setting the "pushed" config values. Java reflection, even when the reflection objects are cached, will introduce new overhead by itself. There are handy libraries built on ASM that provide "reflection" based on byte-code generated classes. One such lib is ReflectASM. Another advantage is that the fname could be replaced with a simple index (https://github.com/EsotericSoftware/reflectasm#avoiding-name-lookup).

I agree that we need to avoid reflection here. The only reflection I was suggesting was on the client-side, where we already use it.

#14 - 01/04/2018 12:27 PM - Greg Shah

> Almost; this is always problematic when building an UI with dynamic widgets - see the calendar. In this case, there are lots of attribute assignments (especially location, size, sensitive, visible and/or hidden), which expose this performance issue easily, as is done in a tight loop.

Is the current GUI application using the calendar control?

We intend to implement a "4GL style" calendar widget to replace this usage, though I wasn't expecting to do it right now.

> Beside the calendar, ADM still relies in certain cases on the same small sub-set of attributes (location, size, visibility and sensitivity) to layout/manipulate the UI.

Yes, this is still an issue, though we will at least make these calls much faster and reduce the overhead.

> The problem we have here is that i.e. HIDDEN, VISIBLE and SENSITIVE are not simple setters - they affect the UI, in more complex ways (see

> the GenericWidget setters for these). But we may need to just move the behaviour directly on client-side...

Understood. Yes, we would need to potentially implement specific (non-generic) down-calls for these.

> As a side note: we are not tracking changed fields, but assigned fields... as I recall it was pretty expensive to interrogate via reflection the previous field value (to send to server side only changed fields). So when ConfigManager.getConfigUpdates is reached, it will have to process all the config fields which were set (and not necessarily changed) by the client-side.

Yes, this probably explains why we see so much activity in the client to server direction. We will need to think about this one as it has real costs that affect performance.

#15 - 01/04/2018 01:28 PM - Constantin Asofiei

Greg Shah wrote:

> Is the current GUI application using the calendar control?
>
> We intend to implement a "4GL style" calendar widget to replace this usage, though I wasn't expecting to do it right now.

The adatesd0.w program is not used so no, we don't need for current GUI app.

#16 - 01/23/2018 07:14 PM - Constantin Asofiei

Hynek, do you recall why you used ConfigManager.duplicate() instead of relying on the backup config instance (from ConfigManager.backupConfigs map), in ConfigSyncManager.registerConfig? Currently at least I see that the 'backup' is not kept in sync on client-side to i.e. reflect the latest server-side state; so I'm inclined to think that, if I make the backup to reflect this, then the duplicate call can be avoided.

#17 - 01/24/2018 02:33 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek, do you recall why you used ConfigManager.duplicate() instead of relying on the backup config instance (from ConfigManager.backupConfigs map), in ConfigSyncManager.registerConfig? Currently at least I see that the 'backup' is not kept in sync on client-side to i.e. reflect the latest server-side state; so I'm inclined to think that, if I make the backup to reflect this, then the duplicate call can be avoided.

The duplicate() is needed to establish a base state for the afterConfigUpdate() to be passed to. This is a short-lived instance needed only during the config synchronization scope (i.e. between CSM.markScopeStart() and CSM.markScopeEnd()).

The reason why backup config is not used is that the config changes in some cases have different granularity than the client-server synchronization round trips.

Are you concerned about performance?

#18 - 01/24/2018 05:24 PM - Constantin Asofiei

Hynek Cihlar wrote:

> Are you concerned about performance?

Yes, but although it gets called many times, it looks pretty fast in the profiler - the duplicate() call doesn't stand out.

#19 - 01/24/2018 06:14 PM - Constantin Asofiei

A state of the point I reached with these improvements:

1. server-side AspectJ should be avoided, especially if it refers to fields in WidgetConfig (and sub-classes); these get assigned during frame setup, and adding this overhead hurts when having a i.e. tight-loop frame creation (i.e. a report which runs another program to output something via a frame and the frame is short-lived, but the program is invoked a lot of times).
2. because of the reason above, although it would have looked a lot nicer to annotated the setter via a i.e. ServerConfigFieldSetter and track all assigned fields during this setter automatically via AspectJ, in the end the solution is to use a pushWidgetAttr(fieldName, value) call instead of pushScreenDefinition(). There are some cases where pushScreenDefinition() can't be removed easily, for example:
   - BaseEntity.setSizeChars and setSizePixels - this can be fixed, I just need to navigate through all the sub-classes and determine which fields need to be pushed
   - some cases in BrowseWidget where fields in both the browse and browse-column get assigned or where other widgets are created (i.e. editor widgets)
   - other cases where widgets are attached/removed in the frame
3. the improvement using pushWidgetAttr way in the DLM report in the regression testing environment is ~10%
4. for ADM case, the feeling when navigating the UI is like is almost the same as the VM.
5. for ADM case, now in the profiler the following stand out:
   - in Web Client mode, the lambda expressions in GuiWebDriver.registerMouseWidgets (and this API) look like a point of interest which can be improved: some lambda expressions are executed millions of times just by navigating the tabs...
   - ServerExports.trigger stands out in both web and swing clients - Eric, were you able to profile the FWD server? I can't attach safely with Oracle JDK 1.8.0_101.
6. with ADM client, it looks like ReflectASM has a similar performance as Java's MethodHandle (I don't see anything noticeable). I plan to run one with ReflectASM changes tonight and another with MethodHandle tomorrow morning - if there is no significant difference, I'll leave the MethodHandle code.
7. an interesting part between pushScreenDefinition and pushWidgetAttr is that in the same amount of profiled time (~4ms), pushWidgetAttr executed ~6000 times and pushScreenDefinition ~1000 times.

The current work was committed to branch 3246a rev 11216 (from trunk 11215). Still needs cleanup and chose either ReflectASM or MethodHandle.

#20 - 01/24/2018 06:23 PM - Constantin Asofiei

Forgot to note:

1. menus are treated via pushWidgetAttr, when feasible
2. windows are treated via LogicalTerminal.pushWindow - old way - as these are kept in a different registry on client-side

#21 - 01/25/2018 11:15 AM - Eric Faulhaber

Constantin Asofiei wrote:

> Eric, were you able to profile the FWD server? I can't attach safely with Oracle JDK 1.8.0_101.

I am always able to attach and do CPU sampling, but if you are asking about true CPU profiling, it doesn't always work. It works most of the time, if I choose short use cases (i.e., those that only last a few seconds with normal use). If I try to profile something long running, it just takes too long and often gets hung up in class instrumentation. I am using OpenJDK 1.8.0_151.

#22 - 01/25/2018 03:20 PM - Eric Faulhaber

Sorry, I forgot to mention two important things I've had to do to get true CPU profiling working in VisualVM:

- upgrade to VisualVM 1.4 (profiling with the 1.3 version that comes with JDK8 never worked for me, though sampling worked ok);
- pass -Xshare:off to the JVM when launching the FWD server.

Even then, it is a bit flaky and doesn't always finish the instrumentation part. When this happens, I have to shut down visualvm and kill -9 the FWD client and server and start over.

#23 - 01/25/2018 08:19 PM - Constantin Asofiei

Hynek Cihlar wrote:

> The reason why backup config is not used is that the config changes in some cases have different granularity than the client-server synchronization round trips.
>
> Are you concerned about performance?

Actually, client-side seems OK, but server-side does not - we are calling ConfigSyncManager.registerConfig for each and every widget instantiation, WidgetConfig registration with ConfigManager and more. As I understand, this would be needed only by afterConfigUpdate (via ConfigSyncManager) - but look how the afterConfigUpdate is used by the server-side:

1. ControlTextWidget, EditorWidget - don't require beforeUpdate
2. FrameWidget - only a case of visibility change
3. WindowWidget - uses realized flag

So I don't understand why server-side requires this, because:

1. @SyncConfigChanges annotation is not using realized/visibility, and is used only in three places:

```
EditorWidget.setHeightCharsWorker(double)
EditorWidget.setWidthCharsWorker(double)
GenericFrame.createFrame(Class<T>, String) <T extends CommonFrame>
```

2. ConfigManager.syncConfigChanges is also used in lots of places which have nothing to do with the config fields used by afterConfigUpdate.

This was also being called on each trip to the server (via ConfigManager.applyConfigUpdates) - I think I can solve this case by using applyConfigUpdate(backup) explicitly.

So, am I missing something fundamental related to SyncConfigChanges on server-side? Because I can't see a good use-case where all this overhead would be needed, just to ensure afterConfigUpdate gets called...

And to summarize, I get ~600k calls for registerConfig on server-side just by navigating some page tabs.

#24 - 01/25/2018 08:22 PM - Constantin Asofiei

Greg, you can do a review for 3246a rev 11220; is cleaned up and what is left is:

1. avoid pushScreenDefinition for size-related attributes
2. the server-side SyncConfigChanges behavior
3. history entries

#25 - 01/26/2018 01:52 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > The reason why backup config is not used is that the config changes in some cases have different granularity than the client-server synchronization round trips.
> >
> > Are you concerned about performance?
>
> Actually, client-side seems OK, but server-side does not - we are calling ConfigSyncManager.registerConfig for each and every widget instantiation, WidgetConfig registration with ConfigManager and more. As I understand, this would be needed only by afterConfigUpdate (via ConfigSyncManager) - but look how the afterConfigUpdate is used by the server-side:
>
> 1. ControlTextWidget, EditorWidget - don't require beforeUpdate
> 2. FrameWidget - only a case of visibility change
> 3. WindowWidget - uses realized flag

The idea behind the sync manager is to reduce the number of calls to afterConfigUpdate() by establishing the so called synchronization scopes and calling afterConfigUpdate() of the touched config owners only when the root scope is finished. The configuration changes on the client may get complex and proved to be hard to manage by hand.

But if there is only a handful of places on server where the config changes do require afterConfigUpdate() and the performance overhead is too much I agree we should not bother.

#26 - 01/26/2018 12:14 PM - Greg Shah

Code Review Task Branch 3246a Revision 11221

This is a really great set of changes! I'm excited to see them finalized and merged to trunk.

I'm especially interested in the use of ReflectASM and the fact that you found some performance benefit over MethodHandle usage (which itself is faster than normal Java reflection). This suggests we may have performance to be gained in other parts of the project (ControlFlowOps and com.goldencode.p2j.net.* come to mind).

I don't see much that is wrong. I just have a few questions:

1. There are some places which directly edit the frame.config in GenericFrame (setDialogBox(), setRetain(), setScroll(), setSideLabels(), setTitle(String title, ColorSpec titleColor), setScreenIO(), setStreamIO(), setUseText()) and in the old code it had a pushScreenDefinition(), but the new code no longer pushes changes to the client. This is on purpose?

2. Same question as item 1, but for MenuItemWidget.setSubTypeInt().

3. There is still some reflection usage in ConfigManager and WidgetConfigDef. Does ReflectASM have potential benefit there?

#27 - 01/26/2018 12:20 PM - Constantin Asofiei

Greg Shah wrote:

> 1. There are some places which directly edit the frame.config in GenericFrame (setDialogBox(), setRetain(), setScroll(), setSideLabels(), setTitle(String title, ColorSpec titleColor), setScreenIO(), setStreamIO(), setUseText()) and in the old code it had a pushScreenDefinition(), but the new code no longer pushes changes to the client. This is on purpose?

These can never be set via an attribute - they are just options set at frame setup, and previous pushScreenDefinition() was already a no-op.

> 2. Same question as item 1, but for MenuItemWidget.setSubTypeInt().

Same as above.

> 3. There is still some reflection usage in ConfigManager and WidgetConfigDef. Does ReflectASM have potential benefit there?

WidgetConfigDef is a one-time setup at client startup. I'm not sure what you mean for ConfigManager - there is no java.lang.reflect usage there. Can you point me to some code you refer to?

#28 - 01/26/2018 12:55 PM - Greg Shah

> WidgetConfigDef is a one-time setup at client startup.

Good point.

> I'm not sure what you mean for ConfigManager - there is no java.lang.reflect usage there. Can you point me to some code you refer to?

I was referring to the custom setter code in CM.setWidgetConfig().

#29 - 01/26/2018 12:57 PM - Constantin Asofiei

Greg Shah wrote:

> > I'm not sure what you mean for ConfigManager - there is no java.lang.reflect usage there. Can you point me to some code you refer to?
>
> I was referring to the custom setter code in CM.setWidgetConfig().

This code is already removed, as WidgetConfigDef.customFieldSetters is no longer needed and was removed, too:

```
      MethodHandle customSetter = wdef.customFieldSetters[fid];
      if (customSetter != null)
      {
         try
         {
            customSetter.invoke(cfg, value);
         }
         catch (Throwable t)
         {
            final String msg = "Could not process custom setter for config attr %s for " +
                               "widget %s with class %s.";
            String fname = wdef.fieldNames.get(fid);

            throw new RuntimeException(String.format(msg, fname, cfg.id, wdef.cfgCls.getName()),
                                       t);
         }
      }
```

#30 - 01/26/2018 01:21 PM - Greg Shah

I only see rev 11221 in bzr. Anyway, the changes are good. Do you want to test it separately from 3435a?

For that matter, it is time to get 3435a into testing too...

#31 - 01/26/2018 01:42 PM - Constantin Asofiei

Greg Shah wrote:

> I only see rev 11221 in bzr. Anyway, the changes are good.

It shouldn't be in 11221 in bzr... I'm not seeing it there, in bzr.

> Do you want to test it separately from 3435a?

I've already been testing it with our runtime-testing set. I need to do more in-depth manual GUI testing, OTOH. So I'd like to avoid merging it in 3435a.

#32 - 01/26/2018 01:44 PM - Greg Shah

> It shouldn't be in 11221 in bzr... I'm not seeing it there, in bzr.

Sorry, you're right. I was looking at the 3435a ConfigManager...

> So I'd like to avoid merging it in 3435a.

That works for me.

#33 - 01/30/2018 05:32 PM - Constantin Asofiei

3246a rev 11225 is the candidate to be merged trunk if it passes testing. What else I have on queue:

1. screenLocation and screenPhysicalLocation info should be cached, but I couldn't find yet an easy way to invalidate these if the widget is moved, re-parented, etc
2. registerMouseWidgets is still slow... I have some changes (mostly to reduce/remove walking the entire parent chain and some other optimisations), but are not enough.
3. there is some slowdown related to static menu registration (excessive calls for pushMenuDescription).

#34 - 01/31/2018 01:44 PM - Greg Shah

Code Review Task Branch 3246a Revision 11225

I'm good with the changes.

> 3246a rev 11225 is the candidate to be merged trunk if it passes testing.

You can merge to trunk if it passes testing.

#35 - 01/31/2018 03:21 PM - Eric Faulhaber

There was one failure in the ETF dev tests; running again in case it was a false negative.

#36 - 02/01/2018 12:37 PM - Constantin Asofiei

3246a rev 11226 fixes an issue for GUI redirected mode exposed by my changes.

If I get the go-ahead from Eric, I can merge it to trunk.

#37 - 02/01/2018 12:41 PM - Eric Faulhaber

The ETF dev tests passed on the third run. Search tests passed on the first run. This was all with revision 11225. How significant is the change in revision 11226? Do I need to re-run with that revision?

#38 - 02/01/2018 12:42 PM - Constantin Asofiei

Eric Faulhaber wrote:

> The ETF dev tests passed on the third run. Search tests passed on the first run. This was all with revision 11225. How significant is the change in revision 11226? Do I need to re-run with that revision?

You don't need to re-run, the change is just for GUI and doesn't affect ChUI.

Is there any concern on ETF dev tests passing on the third run?

#39 - 02/01/2018 12:51 PM - Eric Faulhaber

Constantin Asofiei wrote:

> Is there any concern on ETF dev tests passing on the third run?

This sometimes happens, I'm not sure why. The tests are not necessarily deterministic, as inputs may vary. Different tests failed in the first two runs, and those that failed in one run passed in the other, and all passed in the third.

I think we're ok to merge. I'll run the batch tests for additional coverage, but I don't expect any issues. Of course I'll let you know if there were any problems, but I don't think we need to wait.

#40 - 02/01/2018 01:04 PM - Greg Shah

ChUI regression testing passed?

#41 - 02/01/2018 01:05 PM - Constantin Asofiei

Greg Shah wrote:

> ChUI regression testing passed?

Yes. 3246a was merged to trunk rev 11216 and archived.

#42 - 02/01/2018 01:17 PM - Constantin Asofiei

Is it OK to use 3246b for my other WIP performance issues?

#43 - 02/01/2018 01:50 PM - Greg Shah

Yes, it is OK.

Are the changes risky?

#44 - 02/01/2018 02:00 PM - Constantin Asofiei

Greg Shah wrote:

> Yes, it is OK.
>
> Are the changes risky?

The sub-set I have now is not risky, can be added to 3435a. What I'm planning to add next (cache of parent(), screenLocation() and screenPhysicalLocation()) will be risky.

#45 - 02/01/2018 02:29 PM - Greg Shah

Actually, I would prefer putting it in a different branch. We have so many in process now (3413a, 3444a, 3440a, 1830c) and we need to reduce the churn. 3435a needs to merge to trunk.

#46 - 02/01/2018 02:57 PM - Constantin Asofiei

I'll use 3413a for this set of changes. Will use 3246b for the caching attempt.

#47 - 02/01/2018 05:56 PM - Eric Faulhaber

ETF batch tests passed for 3246a/11225.

#48 - 02/02/2018 09:06 AM - Constantin Asofiei

3413a rev 11250 contains these changes:

- Fixed pushWidgetAttr() for menu cases.
- Improved performance for dynamic menu registration.
- Improved performance for ancestor calculation, by caching the computed values.
- Minor performance improvement for parent() (avoid multiple calls).
- Performance improvements for registerMouseWidgets().

What's left for this is caching the screenLocation() and screenPhysicalLocation() values; another approach is to disable registerMouseWidgets() once it was executed, and re-enable it only when the client gives control to the server-side (i.e. when TC.getChanges() gets called); otherwise, registerMouseWidgets() is a no-op, as only server-side can do something with the UI which might affect the widgets.

But this will just hide the underlying performance of the screen location values.

#49 - 02/02/2018 09:08 AM - Constantin Asofiei

Also, the changes in 3413a/11250 passed runtime testing.

#50 - 02/07/2018 02:30 PM - Constantin Asofiei

The registerMouseWidgets optimizations and caching of screenLocation and screenPhysicalLocation values is almost done - manual testing looks OK, need to perform ChUI regression testing and will commit on top of 3413a once it passes.

From profiling, another issue which stands up is the trigger lookup; as the UI has many triggers (from a very large menubar plus buttons), even when having focus in a secondary window, the trigger lookup still walks the entire list of event definitions (from all windows...). I don't think our simple list is the best way to keep these, we might need a different data structure (maybe some kind of search tree, not sure yet...) to determine if there are triggers associated with a certain key press, mouse event, etc. At minimum, we should separate them on a per-window (plus 'anywhere') basis. But we can do this sometime later.

#51 - 02/07/2018 02:37 PM - Greg Shah

> At minimum, we should separate them on a per-window (plus 'anywhere') basis.

Agreed. This seems like a quick win. The original implementation was done in ChUI where the multi-window environment was not a consideration. As the system stands now, many (or sometimes even most) triggers can be dropped immediately because they are not associated with the active window.

Can you do this quickly?

#52 - 02/07/2018 02:51 PM - Constantin Asofiei

Greg Shah wrote:

> At minimum, we should separate them on a per-window (plus 'anywhere') basis.

> Agreed. This seems like a quick win. The original implementation was done in ChUI where the multi-window environment was not a consideration. As the system stands now, many (or sometimes even most) triggers can be dropped immediately because they are not associated with the active window.
>
> Can you do this quickly?

Not sure. On a second thought, the generic case allows a trigger to be specified for widgets in different windows; and widgets don't know the window until realization. I'll keep this in the back of my mind and if I think I can see a simple solution, will try it.

#53 - 02/08/2018 11:46 AM - Constantin Asofiei

3413a rev 11265 has the changes for caching the screenLocation() and screenPhysicalLocation() values, plus reduce the amount of GuiWebDriver.registerMouseWidgets() calls. Please review.

#54 - 02/08/2018 03:17 PM - Greg Shah

Code Review Task Branch 3413a Revision 11265

Generally, the changes look quite good.

1. Trigger-based recursion is not possible in the layout processing so that is why the thread-local layoutDepth in LocationCacheAspect is safe (it can never be accessed simultaneously on more than 1 thread). Do I understand the logic properly?

2. Is there a reason that the tk.getInstanceDriver().setControl(true) in TC.applyChanges() is not in the finally block?

#55 - 02/08/2018 03:21 PM - Constantin Asofiei

Greg Shah wrote:

> Code Review Task Branch 3413a Revision 11265
>
> Generally, the changes look quite good.
>
> 1. Trigger-based recursion is not possible in the layout processing so that is why the thread-local layoutDepth in LocationCacheAspect is safe (it can never be accessed simultaneously on more than 1 thread). Do I understand the logic properly?

Actually, there's no need of ThreadLocal for layoutDepth, not even AtomicInteger, as this can only be reached on the main thread. I don't know for other threads to be able to call layout.

> 2. Is there a reason that the tk.getInstanceDriver().setControl(true) in TC.applyChanges() is not in the finally block?

No, I'll move it there.

#56 - 02/08/2018 04:13 PM - Constantin Asofiei

Previous note issues were fixed in 3413a rev 11268.

#57 - 02/09/2018 02:26 PM - Greg Shah

- % Done changed from 0 to 100

- Status changed from New to Closed

- Assignee set to Constantin Asofiei

#58 - 02/09/2018 02:26 PM - Greg Shah

3413a was merged to trunk as revision 11219.

Files

profiling_bitset_get_method_usage_20180103.png	369 KB	01/03/2018	Greg Shah
profiling_bitset_get_method_usage_client_20180103.png	370 KB	01/03/2018	Eric Faulhaber