Subject: A problem with UPP application scale . . . UPP exonerated, the rest of the story.
Posted by jlfranks on Thu, 19 Feb 2009 18:41:24 GMT
View Forum Message <> Reply to Message

Our UPP application has been growing for over a year.
We have over 75 screens, custom UPP widget library
designed for touch screen access, virtual keyboard for editing, etc.

The problem is the header and .lay files. Any modification
anywhere causes the world to recompile. Just adding white
space to a header file, or changing a label content (.lay)
causes a long compilation on a dual core machine with 2 GByte
RAM running openSuSE 10.2

Is there a way of breaking the dependencies so that the menu
hierarchy is not so sensitive to recompiling everything?

Can pure abstract base classes be used somehow to present
interface and not implementation to insulate from changes
in headers and .lay files?

Any suggestions are welcome.

--jlf

Subject: Re: A problem with UPP application scale . . .
Posted by Mindtraveller on Thu, 19 Feb 2009 20:41:46 GMT
View Forum Message <> Reply to Message

What header file are you talking about? What do you have inside this file? Please post little spicific details about where and what do you have exactly.
More correct and OOP-style task decomposition will perhaps decrease the number of dependencies. Pure abstract classes will help but it is a sort of thing you should do when decomposition is finally polished and you need to avoid some more dependencies.

Subject: Re: A problem with UPP application scale . . .
Posted by jlfranks on Fri, 20 Feb 2009 16:25:59 GMT
View Forum Message <> Reply to Message

Well, I'm discussing the UI Class header file(s) that make
up a UPP UI application.

Starting at the top of the hierarchy we have a header file that inherits from a layout file (.lay) of type TopWindow.

Note in the example below, the #includes for more UI class
files for next level of hierarchy. This is the problem. Those
files are structured in similar manner to top level except
the inheritance is from their layout files and perhaps of type
ParentCtrl.

Carry this down a four or five more levels until you have a
populated UI Class hierarchy of 70 some odd classes.
Any change anywhere in the layout files or the header files
causes massive re-compilation. Now add multiple UI developers
for a big project and progress becomes grindingly slow.

And, the clue that something is really not right is when the
content of a label is changed at the bottom of the UI class
hierarchy. This should not affect many files, but it does
because the layout file changes. Or, added somemore callbacks
to the implementation file and declare their signature in
the corresponding UI Class header file. Same thing happens,
i.e., compile the world.

There really is no problem for small projects. This is a problem
related to scale.

The UI Classes exhibit the inheritance (is-a) relationship
instead of containment (has-a). The latter has the opportunity
for abstraction by using pointers to UI classes, forward
references, and moving the #includes of other UI classes to
the .cpp implementation file. This header file is less
encumbered with dependencies to the rest of the UI structure.

With the current UPP implementation I cannot figure out how
to break that dependency. The required macros are coupling
other parts of the UI class hierarchy via the layout file
directly into the header files. I've tried for about a day to
come up with some scheme to remove this characteristic but
without success.

Does this make the problem statement clearer?

Examples UI Class header file:

```
-------------------------------
...guards...

#include <CtrlLib/CtrlLib.h>
using namespace Upp;
```

```
// The next includes is where trouble begins. It is
// needed because the layout file includes items to get
// to those next levels of the class hierarch (other screens).
//
#include <...more UI Class files for next level of hierarchy...>
#include <...more UI Class files for next level of hierarchy...>
#include <...more UI Class files for next level of hierarchy...>
#include <...more UI Class files for next level of hierarchy...>
#include <...more UI Class files for next level of hierarchy...>
#include <...more UI Class files for next level of hierarchy...>

#define LAYOUTFILE <here/MyUIClass.lay>
#include <CtrlCore/lay.h>

class MyUIClass : public WithMyUIClassLayout<TopWindow>
{
  public:
    typedef MyUIClass CLASSNAME;
    MyUIClass();

  // ... other stuff for this class ...
};
```

Subject: Re: A problem with UPP application scale . . .
Posted by Mindtraveller on Sat, 21 Feb 2009 13:03:55 GMT

View Forum Message <> Reply to Message

Since U++ uses references, using virtual UI-classes will hardly be handy here.

The first thing which comes to my mind after reading your problem is idea of moving specific UI-details from headers into .cpp files. So changing one UI component will lead to recompilation of corresponding .cpp-s instead of all the project.
Until TheIDE` Layout editor creates only .lay files (which are really .h files) you should stop using it and construct all your UI manually. This code will be inside constructors defined inside .cpp files.

There are 3 ways of improving this situation:
1) Design .lay with Layout editor, then adopt lay-code lines from it into corresponding .cpp inside some ctor.
2) The same as (1) but instead of manual conversion you may write conversion utility which will generate code from .lay statements (this should be rather simple task I suppose).
3) Ask U++ authors on TheIDE wishlist subforum to tune Layout Editor for it to be able to output your design as the code into some labels inside your code instead of .lay file generation.

Subject: Re: A problem with UPP application scale . . . UPP exonerated, the rest of the story

Posted by jlfranks on Tue, 24 Feb 2009 23:26:05 GMT
View Forum Message <> Reply to Message

Well, there is more to the story.

Mis-diagnosis of the problem for a start

True enough, the is-a relationship with inheritance will
cause a UI class file at the bottom of the hierarchy to
re-compile everything in that line all the way to the top,
maybe four or five levels deep. Still not too bad because not
all that many files are involved in a re-compile.

The top-level UI class was providing utility for lower level
classes, such as special callbacks, update of status bar, etc.
So, each lower level file that needed access to those interfaces
included the top-level .h file (the one that inherits from
top-window). That was the cause of the real problem because now
any change anywhere in the hierarchy causes a re-compilation of
any UI class that included the top-level .h, which was nearly
all of them.

The solution was to create a pure abstract base class that
provided the interface. However, it wasn't so pure because the
callbacks (static in nature) had to be put there also. Now the
top level UI also inherits from this interface and implements
the virtual methods that were pure abstract in the base class.
A provision was included for getting access to the pointer to
the base class, which all lower level UI classes can get.

Compile time is now good. UPP was not the problem, but did
gives us the rope to hang ourselves, which we promptly did.

Everything is going well now. Thank you for your suggestions.
I'm going to look at those closely. Those will just make things
that much better. We are over the hump now.

--jlf

Subject: Re: A problem with UPP application scale . . . UPP exonerated, the rest of the story
Posted by Mindtraveller on Wed, 25 Feb 2009 00:44:48 GMT
View Forum Message <> Reply to Message

In this case abstract classes (by the way, you may define pure virtual member with function body) are better than my solution. Because adding new member functions doesn`t cause full recompilation.

My proposal is (and was) about GUI elements only. If you, say, have Map<String,Ctrl> - then you may add GUI elements dynamically inside ctor. So adding any more GUI elements wouldn`t cause rebuild. But it looks rather artificial solution to me.

Subject: Re: A problem with UPP application scale . . . UPP exonerated, the rest of the story.

Posted by mirek on Wed, 25 Feb 2009 09:58:18 GMT

View Forum Message <> Reply to Message

jlfranks wrote on Thu, 19 February 2009 13:41Our UPP application has been growing for over a year.
We have over 75 screens, custom UPP widget library
designed for touch screen access, virtual keyboard for editing, etc.

The problem is the header and .lay files. Any modification
anywhere causes the world to recompile. Just adding white
space to a header file, or changing a label content (.lay)
causes a long compilation on a dual core machine with 2 GByte
RAM running openSuSE 10.2

Well, bad news first, there is a price to be paid for U++ design approach. The design decision is to reduce code complexity via close layout -> class relationsthip and strictly avoiding all kinds of unnecessarry pointers.

Then, to address the recompile problem, we have rather tried to actually make recompile FAST. Using BLITZ, you get compiled 8-16 files for a time you get compiled single one in large project, as a result of changing single label. BLITZ really makes "recompile because of header change" problem next to non-existant.

However, this really is not the end of the story. If you really wish to or need to, you can still keep .lay separated by kind of PIMPL, e.g.:

header:

```
class MyDialog {
   struct MyDialogImp;
   One<MyDialogImp> dlg;

   MyDialog() { dlg.Create(); }
};
```

.cpp :

```
#define  LAYOUTFILE "MyDialogLayout.lay"
#include <CtrlCore/lay.h>

class MyDialogImp : WithMyDialogLayout<TopWindow> {
};
```

or even:

header:

```
class MyDialog : TopWindow {
    struct MyDialogImp;
    One<MyDialogImp> dlg;

    MyDialog() { dlg.Create(); Add(dlg->SizePos()); }
};
```

.cpp :

```
#define  LAYOUTFILE "MyDialogLayout.lay"
#include <CtrlCore/lay.h>

class MyDialogImp : WithMyDialogLayout<ParentCtrl> {
};
```

I hope you got the idea... There are countless variations of this approach...

Personally, I would not do this, because you are sacrificing human work for compile time.

BTW, how many actual lines has the app? 75 screens ain't that many. My computer does full debug mode rebuild of theide (200000 lines of .c/.cpp and 50000 lines of .h/.lay) in about 25 second.

I suspect you are not using BLITZ (you are on ARM, right?), so maybe your first remedy would be to try BLITZ after all. Differences are DRAMATIC!

Speaking about it:

1) I would be interested to hear how well Painter works on ARM, given that it is using a lot of FP.

2) What is the status of your U++ sources? I believe you had to change quite a lot to make it work with your platform, are not you interested in adding some patches in future U++ to make your work easier?

Mirek

Subject: Re: A problem with UPP application scale . . . UPP exonerated, the rest of the story.
Posted by jlfranks on Sat, 14 Mar 2009 20:23:49 GMT
View Forum Message <> Reply to Message

Sorry for the work related delay in getting back.

You explained this well, yes I get the idea.

Regarding lines of code (using sloccount on Linux)
for just UI related stuff . . .

SLOC = 24,741 for the Main UI

SLOC = 8,588 for custom UI libs to handle touchscreen, virtual
             keyboard, skinning, etc.

Regarding BLITZ - no we are not using it. We really did not
know enough about it to feel comfortable with it when we
first got started with UPP a couple of years ago. Now, we can't
use it because it complains about some stuff that we don't
have a clue as to how to fix.

Painter works GREAT on ARM AT91SAM9263, but we use a GNU
cross-compiler that produces EABI code. This implements an
efficient floating point emulation (compared to old method of
processor exceptions for floating point emulation).
All of this has really pretty good performance real-time
displaying UPP gauges, meters, grids, etc.

We have frame buffer interface with DMA update of a color
LCD color display (640 x 480) on a layered fabric inside the
SOC all translating to very fast graphic processing for a
little processor.

We didn't have to patch UPP a whole lot to make things work.
Yes we would be interested in tying in to the latest UPP
sources, and I think that will happen, but not right now.
We are still on 2007.1 and can't move forward because our
custom widgets are specializations of UPP lib hiearchy as
it existed with 2007.1.

The refactoring and performance improvements in your
Ultimate++ reflected in a drastic change to the UPP

class hierarcy and our build would break. We have not had the
time to go back and change our widget library in order to make
use of new UPP. Maybe sometime in the future. We're still
adding features to our product willy-nilly and don't have
a lot of resources to go back and fix stuff to work with
new UPP.

Hope this provides an update that is helpful.

BTW: Just tied our project into Topic++. This is really great.
Only problem is that we won't use it a whole lot since we target
over eight languages and don't have resources to handle all the
translation work that a comprehensive help system would entail.

--jlf

Subject: Re: A problem with UPP application scale . . . UPP exonerated, the rest of the story.
Posted by mirek on Fri, 20 Mar 2009 09:32:57 GMT
View Forum Message <> Reply to Message

jlfranks wrote on Sat, 14 March 2009 16:23Sorry for the work related delay in getting back.

You explained this well, yes I get the idea.

Regarding lines of code (using sloccount on Linux)
for just UI related stuff . . .

SLOC = 24,741 for the Main UI

SLOC = 8,588 for custom UI libs to handle touchscreen, virtual
            keyboard, skinning, etc.

Regarding BLITZ - no we are not using it. We really did not
know enough about it to feel comfortable with it when we
first got started with UPP a couple of years ago. Now, we can't
use it because it complains about some stuff that we don't
have a clue as to how to fix.

Painter works GREAT on ARM AT91SAM9263, but we use a GNU
cross-compiler that produces EABI code. This implements an
efficient floating point emulation (compared to old method of
processor exceptions for floating point emulation).
All of this has really pretty good performance real-time
displaying UPP gauges, meters, grids, etc.

We have frame buffer interface with DMA update of a color
LCD color display (640 x 480) on a layered fabric inside the
SOC all translating to very fast graphic processing for a
little processor.

We didn't have to patch UPP a whole lot to make things work.
Yes we would be interested in tying in to the latest UPP
sources, and I think that will happen, but not right now.
We are still on 2007.1 and can't move forward because our
custom widgets are specializations of UPP lib hiearchy as
it existed with 2007.1.

The refactoring and performance improvements in your
Ultimate++ reflected in a drastic change to the UPP
class hierarcy and our build would break. We have not had the
time to go back and change our widget library in order to make
use of new UPP. Maybe sometime in the future. We're still
adding features to our product willy-nilly and don't have
a lot of resources to go back and fix stuff to work with
new UPP.

Hope this provides an update that is helpful.

BTW: Just tied our project into Topic++. This is really great.
Only problem is that we won't use it a whole lot since we target
over eight languages and don't have resources to handle all the
translation work that a comprehensive help system would entail.

--jlf

Thanks for info. I am looking forward for any eventual ARM related patches (well, in fact, I am also looking forward to summer when ARM based notebooks should be finally available - I am definitely getting one to play with it and U++

Mirek