Subject: true dynamic dispatching with Upp?
Posted by kohait00 on Thu, 18 Oct 2012 07:39:36 GMT
View Forum Message <> Reply to Message

hi all...

i bumped into this one which costs me some head ache, because my curent solution is slow in performance.

is there any nice upp (or any) way to do this in c++?

i have a Elements, that would really like to keep untouched (so many design patterns don't go here, besides beeing not sufficient)..
the Elements are polymorph and should be represented in a View somewhere. For this pupose, i have some representation Editors (Display is not enough here). so i run a list of Elements checking all the Elements and try to figure out, which ElementEditor is fit for it. The check is currently done with dynamic_cast<>, but i want to avoid it (but i fear it is not possible).
This is true dynamic type dispatching, so probably it's the only solution.

Things like Visitor pattern and Strategy pattern wont fit here as far as i can tell (because they imply extending the Element and expect Element to know all possible Editors)..

Strategy pattern would be sort of cached Info/Context stored in Element, of which Element is not aware of. Don't want that either.

So what are you guys telling me here? is a type Map the only Thing besides dynamic_cast<>? using typeid? for best performance?

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

class Element
{
//some base class interface
};

class ElementA : public Element
{
//some concrete implementation of Element
};

class ElementB : public Element
{
//another concrete implementation of Element
};

// the above classes dont mustn't know anything of the following classes
```

```
class ElementEditor
{
// generic Element editing
};

class ElementAEditor : public ElementEditor
{
// specific/additional ElementA editing
};

class ElementBEditor : public ElementEditor
{
// specific/additional ElementB editing
};

//

void EditElement(const Element& e)
{
 //how to dynamicly dispatch to the right Element Editor? truely, depending on runtime type of Element?

 //cant use Visitor pattern (implies extending Element with a Visitor interface, which knows of all Editors, Bad!!!)
 //cant use Strategy pattern (implies extending Element with additional hook to invoke a pluged in editor, is more or less a cache of a once chosen editor dispatching)

 //is dynamic_cast<> option the only one possible?

 if(ElementA* p = dynamic_cast<ElementA*>(&e)) { /*invoke ElementAEditor*/ }
 else
 if(ElementB* p = dynamic_cast<ElementB*>(&e)) { /*invoke ElementBEditor*/ }
 else { /* invoke ElementEditor as fallback */ }
}
```

Subject: Re: true dynamic dispatching with Upp?
Posted by Didier on Thu, 18 Oct 2012 21:15:55 GMT
View Forum Message <> Reply to Message

Hi Kohait,

if you can't change you're elements, maybe you can encapsulate them in a helper class that would manage the editing part:

```
class ElementHelperBase {
  public:
  virtual Element* get() = 0;
  virtual void Edit() = 0;
}

template<class ElementType>
class ElementHelper : public ElementHelperBase
{
  private:
  ElementType& element;  // initialized by some constructor

  public:
  virtual Element* get() { return &element; }
  virtual void Edit() { EditElement(element); }
}

// using function overloading
// you can add an 'EditElement()' function for each type

void EditElement(ElementA& element)
{
  ElementAEditor editor;
  ... do you're stuff
}

void EditElement(ElementB& element)
{
  ElementBEditor editor;
  ... do you're stuff
}
```

and finally you can do:

```
void EditElement(ElementHelperBase& e)
{
  e.Edit();
}
```

No need for dynamic_cast<> any more

This will work, but you need to create ElementHelper classes and instead of keeping track of

'Element's you need to keep track of 'ElementHelper's.

I used somthing close to this in ma GraphCtrl class to manage editing the axis properties depending on axis class type

Hope this idea help's you

Subject: Re: true dynamic dispatching with Upp?
Posted by kohait00 on Thu, 18 Oct 2012 21:23:01 GMT
View Forum Message <> Reply to Message

that's a nice idea..

in fact as far as i can see, there are only 2 options here. either using dynamic_cast, as it's a true dynamic dispatch, or store somehow the editor context with the Element, probably within a Helper wrapper. C++ is strong typed, this hits me now

thanks for helping..

EDIT: i remember to have seen sth nice in Xmlize dispatching there a templated Invoke callback is used, which hides away the storage of the type and it's concrete method... this is something similar in fact

---

Subject: Re: true dynamic dispatching with Upp?
Posted by Lance on Wed, 05 Dec 2012 03:24:30 GMT
View Forum Message <> Reply to Message

It depends on whether the exact type of the Elements that you(or more precisely, your code) receives can be determined at compile time or not. If the answer is yes, template specialization is the fastest way to go.

```
template <class T>
struct ElementEditorFinder;

template <>
struct ElementEditorFinder<ElmentA>
{
   typedef ElementAEditor Editor;
};

template <>
struct ElementEditorFinder<ElmentB>
```

```
{
    typedef ElementBEditor Editor;
};
```

But most likely the answer is no. In that case, Dider's solution is insufficient. (Sorry Dider, in no offence, and again, I might be wrong as I always do . If you code is given a Elemnt * which you don't know the exact class name, how are you going to find the correct ElementHelperBase class from it? The time and path taken would be quite similar to when you find the Editor class directly (without using the HelperBase class).

The reason why it's slow is because you do it in a sequential way, plus repeated dynamic_cast might also be costly. You may work around this by using a map or sorted vector or some other facilities support(log(n) time complexity). If the Elements class hierachy happens to provide a distinct integral ID of each Elements class, by all means, use it as the key, otherwise, use the typeid string. For the value field, you cannot use the Editor but you can use a function pointer to a function that will generate a correct matching Editor for the value field.

It takes some extra resources to build the map (or sorted Vector), but if there are really a lot of Elements classes, it might worth the effort.

HTH,
Lance

Subject: Re: true dynamic dispatching with Upp?
Posted by Lance on Wed, 05 Dec 2012 03:43:37 GMT
View Forum Message <> Reply to Message

Quote:

It depends on whether the exact type of the Elements that you(or more precisely, your code) receives can be determined at compile time or not. If the answer is yes, template specialization is the fastest way to go.

Sorry, this part is BS. If you would know it at compile time, you could write the matching Editor type directly without using any fancy techniques. So it has to be a run-time decision.
