Subject: Architectural and C++/Upp questions
Posted by Xemuth on Mon, 28 Sep 2020 13:44:11 GMT
View Forum Message <> Reply to Message

Hello, I'm currently working to improve the package I introduced here :
https://www.ultimatepp.org/forums/index.php?t=msg&th=111 48&start=0&

To improve it and add new functionnality / modularity I started to write bunch of new class:

One of them is named Context and hold a private Vector of Scene Object. Some of public methods allow user to Create/Remove/Get all the scene object depending on ID (int):

```
Scene& CreateScene(); //Return the fresh created scene
bool RemoveScene(int sceneId);
Scene& GetScene(int sceneId); //Return scene depending on ID
```

2 questions come in my head when reviewing this code
-First one : Is it good to return reference of Scene in CreateScene() ? the fact a Vector is holding Scene mean the reference can get dereferenced at next creation/ destruction, on other hand returning the ID of the scene in CreateScene() force user to call GetScene(int sceneId) (which could lead to the same behavior) May I should use Array instead of Vector and return Reference ? What you think ?
-Second one : Since ID is (in term of human view) less simple to remember, is it a good idea to swap int ID to String name, and use name to identify all Scene ?

The Context class hold one Array of Service.
Service class can be seen as a abstract class : class Service{

```
 public:
  Service();
  Service(Service&& service);
  Service(const Service& service);
  virtual ~Service();

  bool IsActive();

  virtual String GetName() = 0;

  virtual void OnCreation();
  virtual void OnDestruction();

  virtual void BeforeUpdate();
  virtual void AfterUpdate();

  virtual void BeforeActive();
  virtual void BeforeInactive();
```

```
  virtual Vector<Value> ReceiveMessage(const String& message,const Vector<Value>& args =
Vector<Value>());

 protected:
  friend class Context;
  Service& SetActive(bool b);
 private:
  bool active = true;
};
```

Later in my code, I want to inherite class of this Service base to integrate some new functionnality to my codebase. By example, my first test was to create an OpenGL service to render things on screen (it worked). The fact is, my OpenGL service hold mutch more functions than the abstract class Service which force me to do this (in order to call the child class functions):

```
Upp::RendererOpenGLService& service =
ufeContext.CreateService<Upp::RendererOpenGLService>();
/**
      I created the service and get a reference of RendererOpenGLService
      .....Later in the code :
**/
Upp::RendererOpenGLService& service =
static_cast<Upp::RendererOpenGLService>(ufeContext.GetService("RendererOpenGLService"));
```

I really don't like this static_cast, I feel like I could do the job I want a much better way. Can you guys confirm me this static_cast could be avoided in profit of better option ?

Find here a diagram of my service and my context : https://i.imgur.com/2BgUoRa.jpg
---

You may have noticed in service class I have this function :
`virtual Vector<Value> ReceiveMessage(const String& message,const Vector<Value>& args = Vector<Value>());`
Since my Context can hold multiple service, I have integrated a way to communicate with all services without knowing which service is who.

Here is how it work (taken from RendererOpenGLService) :

```
virtual Vector<Value> ReceiveMessage(const String& message,const Vector<Value>& args =
Vector<Value>()){
 if(message.IsEqual("AddQueue") && args.GetCount() >= 3){
  try{
     AddDrawToQueue(args[0].Get<String>(),args[1].Get<String>(), args[2].Get<String>());
     return Vector<Value>{true};
  }catch(Exc& exception){
     Cout() << exception << EOL;
```

```
 }
}
	return Vector<Value>{};
}
```

It work but I feel like, using Vector<Value> and returning Vector<Value> is not the best way I could use to communicate between Services (and it do a tons of cast (I think it is really costly in term of performance)). Any advise to improve and/or a better way a communicating between abstract code ?

Don't see this post as a laziness from me (in term of architecture choice)but much more as a knowledge calling. I plan to spend a lot of time in this project to provide the most dynamic/modulable engine to work in 3D environment in Upp. That's why I want to go on a good choices codebase. If you think my project (the way I'm architecting it) is actually a Gaz factory model which will lead to too complex things, feel free to say.

Thanks for the time you will take to awnser this post.
Have good day

Xemuth

Subject: Re: Architectural and C++/Upp questions
Posted by Klugier on Mon, 28 Sep 2020 20:08:53 GMT
View Forum Message <> Reply to Message

Hello Xemuth,

I am glad you ask :) I like threads like this!

---
1. You could create API similar to Unity. You will need two following classes:
- SceneManager (Context probably not very approrpiate name)
- Scene

In context of SceneManager (context) you could use String as in Unity to identify specific scene. The next think I would change is the use of reference. I would opt for pointer, because right now you can not detect errors in your API. Internally you could store it on heap and only in return situation just & for pointer passing. So, the code would look like this:

Scene* SceneManager::CreateScene(const String& id); // In case of error nullptr is returned - you could use other constructs here as well (optional or upp concepts - read Core tutorial for tips). Fine to return Scene here.
bool SceneManager::RemoveScene(const String& id); // bool is fine for error handling
bool SceneManager::HasScene(const String& id); // nice addition to remove
Scene* GetScene(const String& id); // in context of error - nullptr is returned

---

2.

```
class Service{
 public:
  Service(); // <- For one varialbe not need (I do not see the code, so I may not understand
  Service(Service&& service); // <- Not need (break the rule of 5)
  Service(const Service& service); // <- Not need (breaks the rule of 5)
  virtual ~Service(); // <- fine not need for implementation - just mark it = default;

		// Do you plan to support concrete message set then replacing message with enum make sense here...
		virtual Vector<Value> ReceiveMessage(const String& message,const Vector<Value>& args = Vector<Value>());
```

Backing to static_cast proble. Why not use template method instead and do the cast here - it will be hidden for the user:

```
auto* service = ufeContext.GetService<Upp::RendererOpenGLService>("RendererOpenGLService"));
// Pointer for error handling - nullptr in case of error. I suggest using dynamic_cast here. However, you can not distinguish error type here - whenever it fail on dynamic_cast or fail on finding service name... You could add HasService method that will return bool...
```

Reference:
- https://en.cppreference.com/w/cpp/language/rule_of_three

---
3. Vector<Value> - seems like task for optional not vector. In c++17 std::optional should do the job. In the world of U++ you could return One<Value>. For more information please read - Core tutorial (3.11 One).

Anyway in the example you show - bool should be enough:

```
// override - nice to add this and consider channing the name of the method to OnMessageReceive
// in c++11 = Vector<Value>() could be simplify to {}
virtual bool OnMessageReceive(const String& message,const Vector<Value>& args = {}) override{
 if(message.IsEqual("AddQueue") && args.GetCount() >= 3){
  try{
    AddDrawToQueue(args[0].Get<String>(),args[1].Get<String>(), args[2].Get<String>());
```

```
    return true;
  }catch(Exc& exception){ // Don't like exceptions here :)
    Cout() << exception << EOL;
  }
 }
       return false;
}
```


Generally speaking I do not like exceptions in C++. I enjoy c++17 approach that allows to unpack tuple in one line:

```
auto [service, error] =
ufeContext.GetService<Upp::RendererOpenGLService>("RendererOpenGLService"));
if (error) {
  // Log error... etc...
  return;
}

// Make further processing with service...
```

This is exactly the same error handling available in golang. The difference is that it is the only option there :) In your case simple *service should be enough. Alternatively you could use exceptions...

Reference:
- https://en.cppreference.com/w/cpp/language/structured_bindin g

Klugier

Subject: Re: Architectural and C++/Upp questions
Posted by Xemuth on Mon, 28 Sep 2020 21:50:30 GMT
View Forum Message <> Reply to Message

Hello Klugier, Thanks for your awnser and your time !

1. ---
Quote:1. You could create API similar to Unity. You will need two following classes:
- SceneManager (Context probably not very approrpiate name)
- Scene
Indeed the main idea behind Context was to create a Scene manager ! but yeah looking at SceneManager, it will be smarter to create a SceneManager class and give it to the context. Context must carry services and track time ellapsed.

Quote:you could use String as in Unity to identify specific scene Yeah I think it will be easier.
Quote:The next think I would change is the use of reference. I would opt for pointer, because right now you can not detect errors in your API My code is actually full of exception.

2. ---------------------------------
The main idea behind virtual Vector<Value> ReceiveMessage(const String& message,const Vector<Value>& args = Vector<Value>());
is the fact Service can be everything, by example my openGL Service looks like this : https://i.imgur.com/2BgUoRa.jpg
Many of functions here must be used by user, but in order to allow my gameobjects to be render they must call some function of this service, to do it. They send message the service.
The message must be "AddQueue" and it must have 3 args (name of MasterRenderer, Renderer, RawModel to use which are all String). Since service is all up to the developpers which developpe them, it can be anythings

Quote:Backing to static_cast proble. Why not use template method instead and do the cast here - it will be hidden for the user: Yeah it is the same but it will be the best solution I think

Quote:3. Vector<Value> - seems like task for optional not vector. In c++17 std::optional should do the job. In the world of U++ you could return One<Value>. For more information please read - Core tutorial (3.11 One). What if the receveid message must return multiple Value ?

After all, maybe it would be simpler to retrieve the service reference directly in game object and use it to communicate with the service

---

Subject: Re: Architectural and C++/Upp questions
Posted by Oblivion on Mon, 28 Sep 2020 22:08:14 GMT
View Forum Message <> Reply to Message

Hello Xemuth,

Quote:What if the receveid message must return multiple Value ?

Upp::Value can contain multiple values (ValueArrays, ValueMaps, even raw data).

Best regards,
Oblivion

Subject: Re: Architectural and C++/Upp questions
Posted by Xemuth on Mon, 28 Sep 2020 23:48:38 GMT
View Forum Message <> Reply to Message

Indeed, you are right

Subject: Re: Architectural and C++/Upp questions
Posted by mirek on Tue, 29 Sep 2020 06:46:00 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Mon, 28 September 2020 15:44Hello, I'm currently working to improve the package I introduced here : https://www.ultimatepp.org/forums/index.php?t=msg&th=11148&start=0&

To improve it and add new functionnality / modularity I started to write bunch of new class:

One of them is named Context and hold a private Vector of Scene Object. Some of public methods allow user to Create/Remove/Get all the scene object depending on ID (int):

Scene& CreateScene(); //Return the fresh created scene
bool RemoveScene(int sceneId);
Scene& GetScene(int sceneId); //Return scene depending on ID

2 questions come in my head when reviewing this code
-First one : Is it good to return reference of Scene in CreateScene() ? the fact a Vector is holding Scene mean the reference can get dereferenced at next creation/ destruction, on other hand returning the ID of the scene in CreateScene() force user to call GetScene(int sceneId) (which could lead to the same behavior) May I should use Array instead of Vector and return Reference ? What you think ?

Definitely use Array here.

Quote:
-Second one : Since ID is (in term of human view) less simple to remember, is it a good idea to swap int ID to String name, and use name to identify all Scene ?

Maybe you could also make CreateScene return int id instead. Something like nodes in TreeCtrl...

Mirek

Subject: Re: Architectural and C++/Upp questions
Posted by mirek on Tue, 29 Sep 2020 07:07:35 GMT
View Forum Message <> Reply to Message

Klugier wrote on Mon, 28 September 2020 22:08Hello Xemuth,
In context of SceneManager (context) you could use String as in Unity to identify specific scene. The next think I would change is the use of reference. I would opt for pointer, because right now you can not detect errors in your API. Internally you could store it on heap and only in return situation just & for pointer passing.

Never use pointer if you can use reference. Pointers in general produce heap bugs.

```
auto* service =
ufeContext.GetService<Upp::RendererOpenGLService>("RendererOpenGLService"));
// Pointer for error handling - nullptr in case of error. I suggest using dynamic_cast here. However, you can not distinguish error type here - whenever it fail on dynamic_cast or fail on finding service name... You could add HasService method that will return bool...
```

Do not repeat yourself:

```
auto* service = ufeContext.GetService<Upp::RendererOpenGLService>();
```

That said, Service concept itself feels bit like overengineering. Do you really need that level of abstraction? Cannot you just have RenderOpenGL class or something?

Quote:
Generally speaking I do not like exceptions in C++. I enjoy c++17 approach that allows to unpack tuple in one line:

Exceptions in C++ are superior when used correctly.

Anyway, in this context, you should always ask: Is the error handling even worth it? I mean, can the application meaningfully continue after error? Can it do something reasonable with the fact? If not, then just LOG the error and either end the application with message, or fix it so that it can continue like nothing happened. Set resetable error flag. Maybe add Event<> WhenError so that client code can throw the exception if it needs to.

```
auto [service, error] =
ufeContext.GetService<Upp::RendererOpenGLService>("RendererOpenGLService"));
if (error) {
  // Log error... etc...
  return;
}
```

This is terrible idea (yes I suppose Rust and Golang are going this direction but they have additional language constructs to deal with it. Still makes me doubtful about them). This approach will infest your code with tons of meaningless errorhandling code.

I guess it is the time I have started working on that "U++ design philosophy" article....

Mirek

Subject: Re: Architectural and C++/Upp questions
Posted by Xemuth on Tue, 29 Sep 2020 18:07:55 GMT
View Forum Message <> Reply to Message

Hello Mirek,

Thanks for the sharing of your knowledge !
Quote:That said, Service concept itself feels bit like overengineering. Do you really need that level of abstraction? Cannot you just have RenderOpenGL class or something?

I want service to be external to this the codebase of my project. By example my OpenGL service can be seen as kind of plugin (UFEplugin/RendererOpenGLService) which can be included by the developper to extends possibility of my project. (I plan in further to do some plugin to include Assimp or to support Vulkan rendering, Audio engine etc...) I think the best way to perform this is to give an Interface (here Service class) to user and allow him to write is own service/ component (component is part of game object and it's also an abstract class)

Quote:Set resetable error flag. Maybe add Event<> WhenError so that client code can throw the exception if it needs to. do Upp have a place where an event is used to handle Error ?

Subject: Re: Architectural and C++/Upp questions
Posted by mirek on Thu, 01 Oct 2020 09:43:23 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Tue, 29 September 2020 20:07Hello Mirek,

Thanks for the sharing of your knowledge !
Quote:That said, Service concept itself feels bit like overengineering. Do you really need that level of abstraction? Cannot you just have RenderOpenGL class or something?

I want service to be external to this the codebase of my project. By example my OpenGL service can be seen as kind of plugin (UFEplugin/RendererOpenGLService) which can be included by the developper to extends possibility of my project. (I plan in further to do some plugin to include Assimp or to support Vulkan rendering, Audio engine etc...) I think the best way to perform this is to give an Interface (here Service class) to user and allow him to write is own service/ component (component is part of game object and it's also an abstract class)

I would probably need to know more. I am just saying that I woudl triple-check whether existence of Context

ufeContext.GetService("RendererOpenGLService")

is really necessarry...

Quote:Set resetable error flag. Maybe add Event<> WhenError so that client code can throw the exception if it needs to. do Upp have a place where an event is used to handle Error ?
[/quote]

SqlSession. It really is not Event but installable function pointer (that code is now like 20 years old :), but the principle is the same.
