

Re: client –server interaction over XML supporting multiple protocols

Source: http://coding.derkeiler.com/Archive/C_CPP/comp.lang.cpp/2005-03/0374.html

From: David (*FlyLikeAnEagle_at_United.Com*)

Date: 03/03/05

Date: Thu, 03 Mar 2005 06:53:25 GMT

Hello rdh,

On Wed, 2 Mar 2005 07:29:50 UTC, "rdh" <rohit.dhamija@gmail.com> wrote:

- > *Hi David,*
- >
- > *Thanks for the reply. was really very informative. I am in process of*
- > *analyzing various models to develop the server. My basic requirement is*
- > *that :*
- >
- > *1) Server can be easily ported to UNIX/LINUX OSes, initiallly to be*
- > *developed for Windows platform.*

When portability or cross-compatibilty are part of the initial goals, I think it is best to design with those concepts in mind. I generally presume these techniques even though many projects seem to grow out of toolsets that already exist from a given system. Each methodology has its costs and benefits that we must balance.

The *ix sockets are the most standard across many platforms. The Windows WinInet at just the socket layer is also very compatible with the *ix interfaces. Initialization and termination have mild differences.

Threading and process models will be different on most platforms. Design with that in mind. You may find certain thread classes useful. You can also build your own. When you have the knoweldge it is usually best to presume your custom abstraction can grow. When you have no idea about the abstractions common libraries can be helpful but often never provide an optimal solution on any plattform. This is certainly true of GUI abstractions.

Try to keep the code that may vary between the OSes

separate from your common code. For instance, a good design usually keeps the verification logic in one place and the GUI component in another. The GUI components will be expected to be different for each platform. When the verification logic is pure C++ and common code libraries there will be less need for OS–abstracted versions.

This list could go on forever. It might be helpful to keep a light implementation on *ix just to verify things as you build them under Windows.

Beware of the MFC and Windows libraries. They are not portable and *ix variants may have significant use characteristics that make portable code hard to maintain.

> 2) *Implementation will be done in C++*

C++ is available for many platforms. There may be slight differences in what level of STL is available or other common libraries. These can usually be worked around without much difficulty.

> 3) *The interaction / communication will be done via XML*

That is up to you. You will need to decide how packetized data will be handled. Will XML conversations ever span a packet? If so, how will you write that abstraction? If not, will the XML size constraints significantly hinder the request or response size?

Even with XML, what will the conversation look like? HTTP–like conversations are generally simple request–response with a stateless conversation model. Will you have that model or something a bit different?

> 4) *Server should accept the connection over any of the protocols > connection oriented or connection less.*

Your initialization of each protocol may be a bit different on each OS. The base server can be standardized fairly easily. The threading and synchronization models are what might dictate separate implementations.

> 5) *The data to be transeferred between client–server is very light and > doesnt involve complex heavy data.*

I presume you also will not have heavy or complex processing to service requests. Scalability will also be affected by how many conversations and protocols you plan to have active at the same time.

> *So m in process of evaluating various servers. Can you please suggest a > server model that suits my requirement.*

I'd suggest a simple multi-threaded, multi-protocol server as the base. These will probably need to be different for each platform due to the threading and synchronization models required by each OS.

XML can be rather nasty to properly implement unless you are using a rather restricted definition (which is okay). A library common to your presumed target OSes may be helpful. Xerces(sp?) and other libraries come to mind.

Your core request-response processor will likely be portable C++ with use of the XML library calls that you standardize on.

The STL or common C/C++ libraries may provide most of the other tools that you will need.

Database concepts are available on many platforms if they are needed. Other concepts that you may need are likely to be found on various platforms.

Gluing all of these layers together on all target platforms should not be too difficult a task.

If you start on just Windows and plan to provide another OS later, it may be wise to periodically check for Windows specific concepts and adjust your developments as needed.

Keeping the install, path/file concepts, and any UIs separate for each platform may be helpful.

> *B.W how about RPC mechanism ? Can it be implemented in Windows also ???*

Windows has some RPC functionality built in. Care must be taken when depending on IE for your XML or RPC processing. A different tool must be found under *ix. Depending on each interface chosen there may be significant rewrite on each platform. I've seen several XML tools that approach the general API differently on the same platform. Even under Windows, there are perhaps a dozen ways of processing XML or RPC transactions.

In general, anything *ix can do Windows can do. Each platform has certain APIs and abstractions geared toward solving problems in a different way. What takes Windows and Visual Basic a few lines, may take volumes when done elsewhere. There is a reason for the bloat underneath Windows (and linux). Each has some very powerful tools included. Choosing the common abstractions that you need will probably serve you best.

If you can avoid XML, there may be more optimal ways to handle your conversation needs. I usually presume people know when they require XML and accept its inherent processing requirements. I've seen more than a few teams base products

comp.lang.c++. Re: client –server interaction over XML supporting multiple protocols

on XML and later wonder why certain functions take a huge amount of time or produce seemingly endless amounts of data. Just remember that it may be an option to question certain design constraints or decisions if you have control of all the parts.

> *Awaiting for your reply,*
> *Thanks,*
> *rdh*
>

Sorry for the long diatribe. I should have answered tomorrow when I'd had time to get some much needed rest. I'll probably be off–line tomorrow -- or is that later today?

David