After 17 years is it too late to fix C++ runtime extensibility? [closed]

Asked 15 years, 8 months ago Modified 15 years, 8 months ago

Viewed 696 times

9

It's difficult to tell what is being asked here. This question is ambiguous, vague, incomplete, overly broad, or rhetorical and cannot be reasonably answered in its current form. For help clarifying this question so that it can be reopened, visit the help center.

Closed 15 years ago.

The 1992 to 1993 timeframe was pivotal and fateful for C++.

In the '92/'93 time frame I worked on a plugin architecture for Aldus PageMaker, which was coded in C++. PageMaker was built on a C++ OOP framework called VAMP, which assisted its portability between Mac OS and Windows.

So we tried to use the features of C++ to build a plugin architecture. This proved to be very problematic for C++ classes due to the so-called brittle base class problem.

I proceeded to write a paper that was published in journals and that I presented at OOPSLA '93 in a

reflection workshop. You can read the pdf format of the paper I presented here:

Time Invariant Virtual Member Function Dispatching For C++ Evolvable Classes (Page to the bottom and click *View or Download*)

I also made contact with Bjarne Stroustrup at a Usenix conference in Portland and proceeded to dialog with him for several months, where he championed the issue of dealing with the brittle base class problem on my behalf. (Alas, other issues were deemed more important at that time - for instance, trying to shepard RTTI through approval.)

Microsoft at this time introduced the COM/DCOM system for the Windows platform that was looked on as a viable solution to the problem. C++ could be used as an implementation language for COM via abstract classes used to define COM interfaces.

However, these days developers shun COM/DCOM - and especially ActiveX, that was based on COM. (In 1994 I went on to work at Microsoft and used C++ and COM/DCOM there for the remainder of the decade. I came to conclude that the technology combination was a definite dead-end. I refer to that experience here: Building Effective Enterprise Distributed Software Systems)

In contrast to all this early C++ history, Steve Job's company, NeXT, devised a plugin architecture using Objective C during the early 90s, and that was integral to

the NeXT Step framework. Today that lives on vibrantly in Mac OS X on Apple's computers and important platforms such as the iPhone.

I submit Objective C enabled solving the plugin problem in a superior manner.

The champions of Java will cite factors such as garbage collection as to why it is more productive than C++. I won't dispute that, but Objective C has not had garbage collection until very recently with 2.0. On the iPhone garbage collection is still not available. None-the-less, the OS X plugin architecture has been entirely viable - due entirely to the merits of Objective C style of OOP vs C++ OOP.

Interestingly, Java has been used to build the most pervasive plugin architectures that exist for any platform or software development community. The Eclipse IDE plugin architecture, which by now is practically legendary, as such architectures go, incorporated a few years ago the Equinox OSGi module management layer. J2EE app servers have supported a *plugin architecture* for EJBs for a decade. These days all J2EE app servers of note have likewise incorporated OSGi to manage runtime bindable modules. The Spring Framework and its runtime bindable unit of the Spring Bean has grown to exceed J2EE - primarily on the strength of its system for managing the binding in of the Spring Bean.

The Java community is still struggling to define an official module management standard yet even as things stand,

the Java platform supports software architectures that incorporate plugin capability more pervasively than any other programming platform.

Despite the weaknesses of Java as a language relative to C#, and that .NET already has a stronger module implementation in its .NET assembly standard, Java is still light years ahead in terms of everyday software systems in wide use that incorporate some form of plugin architecture. Strangely, the Java community doesn't even consciously realize that this is the underpinning of their continued success as an enterprise development platform.

I personally regard the brittle base class problem of C++ to be it's most fatal flaw.

After 17 years have passed since this issue was highlighted to the C++ community, and to the creator of C++, is it now too late to address it?

java c++ objective-c plugins extensibility

Share

Improve this question

Follow

edited Apr 5, 2009 at 7:06

asked Apr 5, 2009 at 4:03

RogerV

3,856 ● 4 ● 30 ● 32

3 I don't disagree, and think this would make a great blog post... but this is more of a site for programming questions than essays on programming languages. – Shog9 Apr 5, 2009 at 4:07

Totally agree with Shog9. Why not post this on your blog? – Can Berk Güder Apr 5, 2009 at 4:10

Roger -- fantastic topic for a blog post, not a great question for Stack Overflow, necessarily.. – Jeff Atwood Apr 5, 2009 at 6:12

Related questions

179	Exotic architectures the standards committees care about
94	OSGi, Java Modularity and Jigsaw
3	Resolving Plug-In Dependency on…

Load 4 more related questions