Subject: Pick overloaded by Rvalue?
Posted by bytefield on Tue, 11 Nov 2008 22:44:30 GMT
View Forum Message <> Reply to Message

Hi,
Will pick concept be available with new C++0x standard or it will disappear letting his place to Rvalue references?

Taking the next simple example, it behave like using pick for the returned data of the function, no copy-constructor or = op. is involved.

```
// rval.cpp
#include <iostream>
using namespace std;

class X
{
public:
 X()
 {
  cout << "Def ctor\n";
 }
 ~X()
 {
  cout << "Def dtor\n";
 }
 X(const X& )
 {
  cout << "Def cpyctor\n";
 }
 void Msg()
 {
  cout << "Showing a message\n";
 }
 X& operator=(X& )
 {
  cout << "= op\n";
  return *this;
 }
};

X fun()
{
 cout << "In fun(), before X x;\n";
 X x;
 cout << "In fun(), after X x;\n";
 return x;
```

```
}

int main()
{
 X&& x = fun();
 x.Msg();
 return 0;
}
```

Quote:g++ rval.cpp -o rval -std=c++0x

If that, have MS introduced(or will) rvalues in their compiler?
Seems g++ is ready for rvalue;

I think because rvalues references are implemented in languages they will be faster than pick concept(which still need copy constructor).

Also i don't know how utf-8 strings will be treated in new standard but if they will be then which string implementation will Upp use it's own or one provided by standard? Porting upp to "new C++" will require some effort .

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Tue, 11 Nov 2008 23:29:10 GMT
View Forum Message <> Reply to Message

Unfortunately, the main disadvantage of rvalue references is that they do not compose:

```
struct Foo {
   Vector<int> x, y;
};
```

- such construct would lack auto-generated pick constructor.

IMO, that is the major drawback (here it is simple, but consider nontrivial case with 20 data members... plus, adding any member, you have to check carefully that you have adjusted the constructor too).

(And yes, I have tried to persuade Howard Hinnant to change this, but has not succeeded).

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by cbpporter on Wed, 12 Nov 2008 07:01:24 GMT

View Forum Message <> Reply to Message

Well I/m more interested in this: Uniform initialization and initialization lists. Seems like GCC can handle this.

Subject: Re: Pick overloaded by Rvalue?
Posted by bytefield on Wed, 12 Nov 2008 09:48:03 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 12 November 2008 09:01Well I/m more interested in this: Uniform initialization and initialization lists. Seems like GCC can handle this.

I've tried some examples given there but just few work with my g++(4.3.2). For example initializers for POD types work but for types from std library it doesn't work.
Work for:int i = {1};
double du[] = {2.2, 3.3, 4.4};
Doesn't work for:std::map<std::string,int> anim = { {"bear",4}, {"cassowary",2}, {"tiger",7} };
std::vector<std::string> vstr = {"Hello", "World", "!"};
std::complex<double> z{1,2};

std::char_traits<char>, std::allocator<char> >, int, std::less<std::basic_string<char, std::char_traits<char>, std::allocator<char> > >, std::allocator<std::pair<const

Are these features implemented later in g++, in new versions of g++ from svn?

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Wed, 12 Nov 2008 10:00:54 GMT
View Forum Message <> Reply to Message

cbpporter wrote on Wed, 12 November 2008 02:01Well I/m more interested in this: Uniform initialization and initialization lists. Seems like GCC can handle this.

Well, these are nice, but I really do not understand the energy put into them.

IMO, they do not significantly improve the way how we can write the C++ code.

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by bytefield on Wed, 12 Nov 2008 11:20:14 GMT

luzr wrote on Wed, 12 November 2008 12:00
IMO, they do not significantly improve the way how we can write the C++ code.

Mirek

I think is simple to write:
Upp::Vector<Upp::String> vs = {"Header1", "Header2", "Something else", "and so on..."};

rather than:
Upp::Vector<Upp::String> vs;
vs.Add("Header1");
vs.Add("Header2");
vs.Add("Something else");
vs.Add("and so on...");

First version is typing less.

Now i'm not really happy with what C++ become...
Because new additions to the language it tend to be too "expert friendly" and still lagging behind other languages at "Standard Library" chapter. For example is simple to make programs in Java, C#, Python having unicode support, sql, etc. so we don't need to reinvent the wheel for every platform.
Now it's not enough that i have more than 10 books about C++, I'll have to buy another treating differences between C++98 and C++0x or perhaps Dr. Bjarne Stroustrup will make another 1092 pages book named "The C++ Programming Language - Special Edition 4 - C++0x" Now i think i know why Linus Torvalds doesn't like C++
It's just me or seems that C++ language is going down?
BTW, do you know that the draft C++0x standard it's just 1314 pages about? What lucky we are...

Subject: Re: Pick overloaded by Rvalue?
Posted by mdelfede on Wed, 12 Nov 2008 15:05:44 GMT

Well, I'm back with properties "a-la-Borland", which, being just a shortcut, can handle nicely the rvalue/lvalue problem AND some other stuffs.
I really don't understand why a so simple construct can't be introduced in c++.

```
struct MyArray<T>
{
  ..............

  property T operator[](int i) {read = readArr, write = writeArr}

  T readArr(int i) { checkbounds(i) ; return buffer[i]; }
```

```
  void writeArr(int i, T d) { copy_if_needed(i); buffer[i] = d;}
};

MyArray<int> arr;

arr[10] = 2;          (uses writeArr())
int i = arr[2];       (uses readArr())
```

Max

Subject: Re: Pick overloaded by Rvalue?
Posted by Mindtraveller on Wed, 12 Nov 2008 18:34:24 GMT
View Forum Message <> Reply to Message

bytefield wrote on Wed, 12 November 2008 14:20 It's just me or seems that C++ language is going down? It`s a pity but C++ is "smallest evil". It is just still powerful and effective. Some time ago it was discussion of C++ alternatives, and it looks like it is nothing as effective and powerful. Yes, we had discussion about D, but IMO it is still no match in cases of efficiency. So for now we have no adequate alternatives...

Subject: Re: Pick overloaded by Rvalue?
Posted by captainc on Wed, 12 Nov 2008 19:27:26 GMT
View Forum Message <> Reply to Message

```
Upp::Vector<Upp::String> vs;
vs.Add("Header1");
vs.Add("Header2");
vs.Add("Something else");
vs.Add("and so on...");
```
Can be written with overloaded operator<< and operator|
Like so:
<< represents .Add() method

```
Upp::Vector<Upp::String> vs;
vs << "Header1" << "Header2" << "Something else" << "and so on...";
```
or
| represents .AddPick() method
```
Upp::Vector<Upp::String> vs;
String hello("hello"), world("world");
vs | hello | world;
```

So we could probably do something like this for a one-liner:

Upp::Vector<Upp::String> vs = Upp::Vector<Upp::String>() << "Header1" << "Header2" << "Something else" << "and so on...";
though I haven't tested this yet.

Subject: Re: Pick overloaded by Rvalue?
Posted by bytefield on Wed, 12 Nov 2008 19:33:43 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Wed, 12 November 2008 20:34bytefield wrote on Wed, 12 November 2008 14:20 It's just me or seems that C++ language is going down? It`s a pity but C++ is "smallest evil". It is just still powerful and effective. Some time ago it was discussion of C++ alternatives, and it looks like it is nothing as effective and powerful. Yes, we had discussion about D, but IMO it is still no match in cases of efficiency. So for now we have no adequate alternatives...

Well, it's still in top and that's matter. 3rd position is quite good for a language with such complexity and i think it will remain there(in top 3) for some years from now on.
I was talking with a colleague about C++ complexity and for those who know C++ new improvements added by new standardization aren't difficult to learn but just think at amount of information which have a beginner to learn, one who is starting with C++ from the ground up.

Quote:Some time ago it was discussion of C++ alternatives, and it looks like it is nothing as effective and powerful.
Have you tried OOP with C? I've read some papers that doing OOP is accessible also to C programmers but i wouldn't recommend doing that - take for example gtk+, it's object oriented but it's very hard to create a new widget in it without doing something wrong.

Subject: Re: Pick overloaded by Rvalue?
Posted by bytefield on Wed, 12 Nov 2008 19:37:27 GMT
View Forum Message <> Reply to Message

captainc wrote on Wed, 12 November 2008 21:27
<< represents .Add() method

Upp::Vector<Upp::String> vs;
vs << "Header1" << "Header2" << "Something else" << "and so on...";

Ok, that's fine but think that you have to overload operator << for every container you have. Having initialization as in example specified by cbpporter let you using it with any type possible so we don't have to code for it.

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Wed, 12 Nov 2008 20:54:01 GMT
View Forum Message <> Reply to Message

bytefield wrote on Wed, 12 November 2008 06:20luzr wrote on Wed, 12 November 2008 12:00
IMO, they do not significantly improve the way how we can write the C++ code.

Mirek

I think is simple to write:
Upp::Vector<Upp::String> vs = {"Header1", "Header2", "Something else", "and so on..."};

rather than:
Upp::Vector<Upp::String> vs;
vs.Add("Header1");
vs.Add("Header2");
vs.Add("Something else");
vs.Add("and so on...");

First version is typing less.

Sure, that is true, however, how often do you have to write this?

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Wed, 12 Nov 2008 21:04:27 GMT
View Forum Message <> Reply to Message

bytefield wrote on Wed, 12 November 2008 14:37captainc wrote on Wed, 12 November 2008 21:27
<< represents .Add() method

Upp::Vector<Upp::String> vs;
vs << "Header1" << "Header2" << "Something else" << "and so on...";

Ok, that's fine but think that you have to overload operator << for every container you have. Having initialization as in example specified by cbpporter let you using it with any type possible so we don't have to code for it.

Well, AFAIK, you need to add specific methods as well.

BTW, you can write it even shorter:

Vector<String> vs = Vector<String>() << "Header1" << "Header2" << "Something else" << "and so on...";

As for C++ being evil/good/bad etc... I like C++ because

- there is no other language covering all areas from HW up to highest levels of abstraction

- this is the only language with destructors

- many alternatives lack efficient operator overloading

- templates are hard to learn, but are the most powerful, esp. w.r.t. efficiency

but there are some downsides too

- as it is pretty complex language, it is hard to get compiler right (fortunately, this mostly seems to be resolved now). Also, with preprocessor, it is hard to get development tools like Intellisense working well (but we have managed, right?

- dynamic library APIs are difficult to maintain if you wish to use the real power of language

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by unodgs on Wed, 12 Nov 2008 22:46:51 GMT
View Forum Message <> Reply to Message

luzr wrote on Wed, 12 November 2008 16:04Intellisense working well (but we have managed, right?

It's quite good, but there are some problems and regression bugs. I'll post a list of my problems soon.

BTW: Qt created QtCreator ide. It also has intelisense, but I didn't test it too hard yet. Link http://labs.trolltech.com/blogs/2008/10/31/qt-creator-tech-p review-released/

Subject: Re: Pick overloaded by Rvalue?
Posted by captainc on Wed, 12 Nov 2008 23:20:43 GMT
View Forum Message <> Reply to Message

Quote:- there is no other language covering all areas from HW up to highest levels of abstraction

- many alternatives lack efficient operator overloading
These 2 reasons are both good and bad. Much confusion comes from having operators do completely different things among different classes. In order to understand C++ code you didn't write yourself, you will have to read the entire codebase before you get a really good handle on what is going on. I think C++ code is the most difficult to maintain for this reason. Someone could

have overloaded a simple operator globally and you are stuck looking at code expecting it to do one thing and not understanding why it is doing something else. If C++ code is not well documented, people will give up and write from scratch!

Quote:- templates are hard to learn, but are the most powerful, esp. w.r.t. efficiency
Conceptually, templates are easy. But in implementation, there are so many rules surrounding them, especially the differences between compilers and how they handle templates, that it is difficult to master them.

I compare C++ to the English language; there are so many exceptions to the rules, that mastering the language cannot be accomplished in a short period of time.

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Thu, 13 Nov 2008 13:37:20 GMT
View Forum Message <> Reply to Message

captainc wrote on Wed, 12 November 2008 18:20Quote:- there is no other language covering all areas from HW up to highest levels of abstraction

- many alternatives lack efficient operator overloading
These 2 reasons are both good and bad. Much confusion comes from having operators do completely different things among different classes. In order to understand C++ code you didn't write yourself, you will have to read the entire codebase before you get a really good handle on what is going on. I think C++ code is the most difficult to maintain for this reason. Someone could have overloaded a simple operator globally and you are stuck looking at code expecting it to do one thing and not understanding why it is doing something else.

I keep hearing this operator overloading argument all over again, but I am still unconvinced.

IMO, you can give wrong names to methods in Java and have the exactly same problem.

E.g. if you see something like

x.Put(y);

in Java, you have as much info as seeing

x << y;

Quote:
If C++ code is not well documented, people will give up and write from scratch!

No pun intended, right? ... RIGHT?

Quote:
Conceptually, templates are easy. But in implementation, there are so many rules surrounding them, especially the differences between compilers and how they handle templates, that it is difficult to master them.

Well, but that is the compiler problem I have mentianed too, is not it?

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Thu, 13 Nov 2008 13:38:52 GMT
View Forum Message <> Reply to Message

Quote:
It's quite good, but there are some problems and regression bugs. I'll post a list of my problems soon.

Check uppdev/Parser1-3 - there is where I store all "A++ bugs". Check the 'format' and try to prepare something similar if possible (e.g. uppdev/ParserUno1).

Thanks

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by cbpporter on Fri, 21 Nov 2008 13:17:45 GMT
View Forum Message <> Reply to Message

Here is a quick video overview of C++0x features: YouTube. The most important part is that that it won't be fully ready by 2011 as an ISO standard .

Subject: Re: Pick overloaded by Rvalue?
Posted by captainc on Fri, 21 Nov 2008 18:48:31 GMT
View Forum Message <> Reply to Message

Yeah, I saw the date and was amazed that it would take till then. I thought they were not going to get features in because the release date was earlier. I don't know how I feel about that late date. I was excited to see C++ take its next step sooner.

In all, this just validates the need right now for a library like U++; Especially with its alternative approaches in the core libraries.

Though, some thought should be put in to how U++ will change when the new standard comes out.

Subject: Re: Pick overloaded by Rvalue?
Posted by cbpporter on Sat, 22 Nov 2008 09:04:36 GMT
View Forum Message <> Reply to Message

captainc wrote on Fri, 21 November 2008 20:48Yeah, I saw the date and was amazed that it would take till then. I thought they were not going to get features in because the release date was earlier. I don't know how I feel about that late date. I was excited to see C++ take its next step sooner.

In all, this just validates the need right now for a library like U++; Especially with its alternative approaches in the core libraries.

Though, some thought should be put in to how U++ will change when the new standard comes out.
Yes, this is not how the situation was presented to the public. After jocking for years "C++0x, where x will be decimal we hope, not hexadecimal", and with the added requirement of submiting the draft in 2008 so it can be acepted by 2009, you would think that since they are on schedule, we would have the draft in 2009. But no, after 2009 there are going two be two years to work out the bugs. So we will get the final standard pretty late.

And until we get compiler support much time will pass. Good news is that we will probably get to play around with some features sooner. By 2009 the quite limited list of C++0x features supported by GCC should have an addition or two. And by 2010, 2011 we should have some support for the uncorrected draft. Question is how portable is that going to be. I think MSC will not experiment the same way GCC does and will try to bring out a mostly complete C++0x compiler somewhere in the future, which personally I hope it doesn't. I sure love MSC, not because it is from MS or anything,just because it's good and fast (and IMO a whole lot better than GCC performance wise; it's not just like MSC is 5% faster or some small number like this), so a C++0x MSC compiler would be a good thing. But I would like them to not give attention to anything else except .NET. I'm hoping even for a Windows version which has a .NET with fewer limitations and without WinAPI. On the other hand, GCC is still great for a production tool (especially when you don't have comparison). If they provide a compiler with uniform initialization, auto, concepts and rvalues, I will be sure to use it, even if is not 100% stable, and I'm sure others will use it too.

But I personally consider C++ doomed. And the final stab that killed it is C++0x. Not that C++0x is not good. It is actually so good that I wish it would have been brought out as C++03. But it's not. And with all those new good features plus good features from previous standards, if you don't have compatibility problems with older software, I'm sure you will be able to use C++ for some time now. But with the rest of the more ugly features, C++ is extremely bloated. What is going to happen after C++0x? TR1? Sure. Add modules to C++ and I'll shut up and probably be able to use happily C++ until I retire if I so desire. Add transparent GC and prove to the world that the performance penalty is not worth getting excited over. Add a really rich MT library (C++0x's is not that rich, only standardized)? Great!!! All these great feature, what could go wrong. Well the bloat can go wrong. Even if you use only the new features, you are going to have to have working

knowledge of all features and this is where the amount of bloat will show it's ugly face. And TR1 is as far as you can go. You can't really bring out a new standard later, because of the bloat. A new programmer will have to learn C++ for years without end, and if the programmer hopes to reach the level of knowledge of C++ veteran switch have gone through each iteration of the language, than a time investment equivalent to their's will be necessary. You will probably have problems with hiring a graduate to work on a C++ project which uses a lot of features. That is true Today. I know a lot of programmers, but few can handle C++ and even less desire to. On the other hand, if you use C++ as C with classes, only use references as const & parameters, anybody can handle C++ and bring a project from start to finish. And that is happening a lot around me.

As for how this will affect U++, it is very hard to tell. First let's consider how much will U++ evolve until then. Personally I think it will remain pretty much the same. I hope will have a moment where we clean it up a little, make it even more modular and remove deprecated stuff. I think we should embrace a 3 level/release approach for deprecation. In the current release we find something that must be deprecated, we move it to level one and we keep it there in next release. Once we have reached that release, we move it to level 2, and keep it there for the next release. Level 2 is not activated by default,so you have to add a flag. To give an example, we find deprecated stuff in 2008.1. While working on the between release versions, we mark it as level one. So in 2008.1 and in 2009.1 we still have the deprecated features and can only be removed by an explicit flag if a user desires. This gives two years for the feature: one of pre-releases, dev versions,and one real release. From the moment 2009.1 is out, level1 items move to level 2, which is not activated by default and new deprecated items move to level one. This leaves you in the situation that a dev after a release removes deprecated items, but that's what dev should do: they can break compatibility. And level two items can always be turned on by a simple flag. And with 2010.1, new items come to level 1, level 1 becomes 2, and level 2 becomes level 3. Level3items are removed.

There are alot of variations on this scheme if it seems to complicated.

Also new packages are sure to surface. Also, I except a larger user base. Not large compared with other toolkits, but large enough that we will have troubles resolving all the requests and support on this forum. The problem could be easily solved with a couple of additional Mireks .

Subject: Re: Pick overloaded by Rvalue?
Posted by unodgs on Sat, 22 Nov 2008 09:49:01 GMT
View Forum Message <> Reply to Message

VC from VS2010 already supports rvaluses, lambdas, auto, and static_assert. But I agree that c++ is too bloated and it is going to be more bloated. This is not good. I would like iso guys to remove headers instead of adding half of those new features. Right now I use c# in my job. I must admit it's a very good language. Even without all those cool features like linq or auto genrating conctructors I found the code written in it a way easier to read (I'm comparing to typical c++ code). And the compilation times are very good. You know I'm tired watching these magic signs like *, &, ->, :: (including combinations) all the time. Switching between header and source simply drives me crazy Now additionaly we have &&... Writing inelisense for c++ is a horror (ask Mirek)
On the other hand I find Upp library marvelous and realy simple to use. What a pity we don't have a better language under it. D is interesting but it follows GC path. GTK fellows wrote VALA maybe we could do something similar like UL (Ultimate language).

PS: You wrote the longest post in history of this forum I guess .

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Sat, 22 Nov 2008 12:19:39 GMT
View Forum Message <> Reply to Message

unodgs wrote on Sat, 22 November 2008 04:49
Writing inelisense for c++ is a horror (ask Mirek)

It is funny challenge for sure

unodgs wrote on Sat, 22 November 2008 04:49
we could do something similar like UL (Ultimate language).

I believe, in a sense, we already did...

You are not going to meet many of those complex constructs in U++ production code.

Anyway, maybe I am just too self-satisfied, but personally I see nothing particulary interesting or helpful in next C++ standard - except maybe clarifications of MT issues, but that is more about what code is "correct by standard", not a game changer in existing practices.

I can imagine a lot of evolutionary improvements and some inevitable cleanups, but overal, for me, U++ succeeded. This is THE environment I wanted to work in

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Sat, 22 Nov 2008 12:22:08 GMT
View Forum Message <> Reply to Message

Well, I guess I should have made it a little bit more clear:

For now, I am going to ignore the next C++ standard.

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by unodgs on Sat, 22 Nov 2008 13:54:11 GMT

View Forum Message <> Reply to Message

Quote:I am going to ignore the next C++ standard
I think variadic templates can improve our callbacks, althoug I don't use more than 3/4 parameters

Subject: Re: Pick overloaded by Rvalue?
Posted by captainc on Sun, 23 Nov 2008 19:53:25 GMT
View Forum Message <> Reply to Message

But I would like them to not give attention to anything else except .NET. I'm hoping even for a Windows version which has a .NET with fewer limitations and without WinAPI.

You are saying you want Microsoft to improve and concentrate on the .NET library?
.NET really gets to me, mainly because it is not cross platform. Porting a .NET application to another platform requires either Mono or an entire rewrite of that code.

To summarize the C++ issue...
The problem with C++: Bloat and complexity.
The benefit of C++: High level abstractions that compile to machine code, full object oriented language.
The problem with Java/C#/etc.: They require an interpreter and thus will always be slower/require more memory than C++ equivalents.
The benefit of Java/C#/etc.: Less complex, less number of exceptions, simpler learning curve, object oriented language, better tool support (Ie. easier to parse).

So what we need is something that compiles to machine code without the complexity of C++.
What is there now? What is coming up that merges the benefits of both C++ and interpreted languages?

Subject: Re: Pick overloaded by Rvalue?
Posted by cbpporter on Mon, 24 Nov 2008 00:18:47 GMT
View Forum Message <> Reply to Message

captainc wrote on Sun, 23 November 2008 21:53
The problem with Java/C#/etc.: They require an interpreter and thus will always be slower/require more memory than C++ equivalents.

No, not an interpretor. Just in time compiler. I am a hobbyist compiler technology nerd, and I think that at least .NET just in time compiler can generate code comparable with native compiler. It is theoretically the same optimizer that works on the internal tree, but this time the tree is expressed in CIL. The performance difference comes from start up JIT time, start-up huge library dynamic loading and of course GC and abundance of references and deeper indirection level than in C++ code. If we were to use GC in C++ and program in a style that uses the same number of references and level of indirection, I think C++ would be as slow as C#. But I think that C# is close to the borderline where it is not considered slow. The problem is that it is still to slow to to write a

Linux kernel in it and it also lacks some features to become a system programing language. And there is no way I can use it with a clear conscience on a 450Mhz machine, where my optimized C++ code blazes by and uses 1/5 of the memory consumption as the equivalent C# app.

Subject: Re: Pick overloaded by Rvalue?
Posted by bytefield on Mon, 24 Nov 2008 05:55:35 GMT
View Forum Message <> Reply to Message

I think if C# will have a JIT compiler which will generate(once on a platform) CIL code mixed with native code for parts of code which run often and need performance it will be the choice for applications (non-system) instead of C++. Just think, only first run of the program on that platform will be slow then it should run from a cache and have the performance almost equal with native programs.

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Mon, 24 Nov 2008 07:32:03 GMT
View Forum Message <> Reply to Message

bytefield wrote on Mon, 24 November 2008 00:55I think if C# will have a JIT compiler which will generate(once on a platform) CIL code mixed with native code for parts of code which run often and need performance it will be the choice for applications (non-system) instead of C++. Just think, only first run of the program on that platform will be slow then it should run from a cache and have the performance almost equal with native programs.

cppporter is right about indirection levels.

E.g. you cannot implement anything close in performance to Vector<String> in C# (ok, String is fundemantal in C#, but imagine you would want to implement yours - you would IMO fail).

BTW, GC makes it faster than if would things be implemented using smart pointers. Smart pointers are generally considered slower than GC. OTOH, GC will eat more memory, at least 30% more if you want any reasonable level of performance.

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by mr_ped on Mon, 24 Nov 2008 08:05:14 GMT
View Forum Message <> Reply to Message

I think as long as you stay "KISS" with your SW, C++ will be always superior in terms of performance.
If you let your code grow in complexity, it may be that "higher" language like Java/C# would help

cut some complexity out, and protect you from some bloat, so the performance of C# vs C++ would be similar then. But that's not C++ failure or C# win, that's solely failure of code design.

Quote:So what we need is something that compiles to machine code without the complexity of C++. What is there now? What is coming up that merges the benefits of both C++ and interpreted languages?

The only thing forcing to use all the complexity of C++ are the underlying libs. If you use simple enough libs, or well encapsulated and well working, so you don't have to dig into them, then just your source decides the level of C++ complexity you have to manage. I mean, you can write very simple (and dumb) source in C++. But it may get quite larger/bloated, than the same functionality written with more complex C++.

Then at some point you have to adopt some source from someone else... and there you go, a) it's easier to write code, than read, b) he will certainly use some C++ features, with which you are not familiar enough and don't feel comfortably using it.

Also I think big part of that "C# is less complex" is strong platform API which helps you more times than C++ stdlib. Surely it's the language definition too, but when I write some simple GUI app with U++, it looks insanely simple. Thank to underlying U++ lib which hides lot of complexity.

Subject: Re: Pick overloaded by Rvalue?
Posted by bytefield on Mon, 24 Nov 2008 09:47:10 GMT
View Forum Message <> Reply to Message

So, is better for Upp and for us to stay away from C++ new standard.

luzr wroteUnfortunately, the main disadvantage of rvalue references is that they do not compose:

```
struct Foo {
Vector<int> x, y;
};
```

- such construct would lack auto-generated pick constructor.
I think that is solved by using rvalue references + "move" constructor, that if every object implement move constructor(thought not auto-generated).

Anyway, why take so much to implement C++0x? I see g++ advancing pretty fast while implement new standard, e.g. now it support rvalue references in 4.3.2 (with -std=c++0x) and initializer lists in g++ from svn. However, is better implmentig it slowly, it means i will have 3 years from now to learn new standard and i think 3 years are enough.

Subject: Re: Pick overloaded by Rvalue?
Posted by mirek on Mon, 24 Nov 2008 10:59:21 GMT
View Forum Message <> Reply to Message

bytefield wrote on Mon, 24 November 2008 04:47So, is better for Upp and for us to stay away from C++ new standard.

luzr wroteUnfortunately, the main disadvantage of rvalue references is that they do not compose:

```
struct Foo {
Vector<int> x, y;
};
```

- such construct would lack auto-generated pick constructor.
I think that is solved by using rvalue references + "move" constructor, that if every object implement move constructor(thought not auto-generated).

Last time I have checked, compiler never generates implicit move constructor.

Moreover:

```
struct Foo {
  Vector<int> x;
  int y;
};
```

Here U++ has natural pick copy of Foo - created by mixing 'deep' and 'pick' constructors of various members.

Well, AFAIK, authors of r-value references considered such feature "too complex and error-prone". Go figure

(It is however possible that the stance has changed since then - in that case, my stance w.r.t. next C++ would completely changed

Mirek

Subject: Re: Pick overloaded by Rvalue?
Posted by captainc on Mon, 24 Nov 2008 18:25:57 GMT
View Forum Message <> Reply to Message

Yeah, I meant JIT Compiler, but for some reason I wrote interpreter.

Quote:Surely it's the language definition too, but when I write some simple GUI app with U++, it looks insanely simple. Thank to underlying U++ lib which hides lot of complexity.
Exactly why U++ is so good... Much cleaner and simple code with good libraries.

I wonder if it would be easy to integrate U++ into existing C++ projects or development environments.
