

# Re: multiple inheritance

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2006-04/msg00241.html>

---

- *From:* "Lebesgue" <nospam@xxxxxxx>
  - *Date:* Mon, 3 Apr 2006 15:46:33 +0200
- 

Shawnk,

thanks for your input. I agree with your arguments and would be glad to see the rewritten proof. If I did my estimation right, then I think I have a proposal which will improve the design "towards shared state space". Looking forward to hear from you

Lebesgue

"Shawnk" <Shawnk@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message <news:C47BFB86-F473-46E3-8FE4-B8C5D007091C@xxxxxxxxxxxxxxxxxxxx>

Lebesque,

Thanks for your patience with the length of my response. Loved the proposed approach which (for brevity) I'll call the PI (Pseudo Inheritance) design technique. I recoded and experimented with this approach this weekend.

Comparing 'PI' to 'MI' (after coding) my executive summary (short form) of the differences as follows.

-----

1. Top level specification of state space in multiple inheritance
  - A. The PI design technique is a wrapper for functionality around pre-existing state space.
  - B. MI allows (obviously) state and functional (static and dynamic) inheritance
  - C. The PI technique offered (by Lebesque) is a 'functional inheritance' (dynamic component) high cost approach.
  - D. The approach fails if the inherited state space (static component) is intrinsic to the target functionality.

Re: multiple inheritance

2. Therefore – (e.g. Child\_cls is Parent\_cls and "kind of is" House\_cls).

A. Dynamically true

B. Statically false

C. Which is a formal agreement of your phrase '(e.g. A is C and "kind of is" B)'

B)'

D. The 'kind of' is a logical marker for the PI non-inheritance of functional state space.

-----

Since I use MI for 'top level specifications' (as above) my use of PI would be as a 'functional wrapper'. As a 'functional wrapper' on pre-existing state space its a good approach if all you needed was a small wrapping functionality around a pre-existing state space.

I have a follow up to this (post) if it is of interest. It has your code redone to prove the 2.A, 2.B assertions above. The proff also shows the severe limitations of C# as a strategic migration tool to factor out common code. It brings out in code the 'kind of' (see above) inheritance architecture flaw in C#.

To restate: The re-write of your code shows the 'statically false' assertion in your 'e.g' given in executive summary above. Sort of explains the above executive summary in code.

Thanks for your thoughtful and insightful input. I haven't used this approach but to overcome C# limitations (MI in IMHO ) this approach is pretty handy :-)

Shawnk

PS. Let me know if you want me to post the rewritten proof of your code.

PPS. High cost based on quantitative/qualitative metrics (1) character count, (2) visual complexity of pass through code and (3) logical complexity of pass through code.