

Why always about syntax??



Deus 12 posts since

Jan 3, 2005

Hi everyone.

I just logged on for the first time, and I'm researching this LOP concept for the last two weeks only, yet, I have some questions, after reading this whole discussion board.

One thinking thread in my brain is stuck in the following loop: if DSLs are constructed to serve their purpose and **ONLY** their purpose (which is reusable), then, their purpose should be strongly narrow, am I right?

As opposed to general programming languages, which have broad usages, you wouldn't need such generic representation of the program as text is.

So if they are (narrow), then they don't need text representation at all, right? I mean, that can be an option, but mainly, if we stick to narrow DSL, an editor can be, as Sergey said, almost anything.

The point is, how big problem would be to learn a DSL, if it has strongly specialised editor just for itself? I mean, wow, it must be a **REALLY** hard job adapting myself to an editor where I can simply draw an user interface (like in VB) and select events, then attach DSL to event actions and simply write mathematic formula to calculate something. This is the concept Sergey was explaining, I suppose.

I understand that the starting point (defining root DSLs) would be time-consuming, but collecting class libraries to suit your needs also is. So there's no difference. Instead of programming class library you do DSL or two, and the next time new problem arise, you construct new DSL out of existing ones.

Why always about syntax??

As the time goes, you could construct DSL in hours.

As I already wrote an email to Sergey (to which he never replied), I said I develop bussines apps today, but pursue game programming, and I can see a lot potential in LOP, mainly because more work could be redistributed to other working units (ie. leaving definition of physics properties to model designer), but that's only the top of the iceberg.

Anyway, I would like some feedback, pointed at the fact that you all stopped at text (cell) editors, and not specialised editors, and continue to argue about learning curve of human-adapted editor.



[Maarten Hazewinkel](#) 181 posts since

Feb 19, 2003 1. **Re: Why always about syntax??** Jan 5, 2005 2:38 PM

I guess the reason we don't go further than a cell/text editor is familiarity. A cell/text editor is just a short step away from the familiar text editors we use all-day.

A second reason is that more specialised editors are harder to create and stand further away from the abstract syntax tree representations. A cell/text based editor is so close to that representations that it could be generated (nearly) automatically.

In many DSL applications, more specialised editors will be appropriate (even required) for domain-oriented people to effectively work with the system. Obvious kinds of editors in business situations are data-flow and work-flow types of editors. Many other types of editors will be invented, but they'll have to grow out of applying LOP and DSLs to specific areas, interacting with domain people to find the best representations.

We're starting a command-line era of LOP, but already knowing that we'll have to evolve to a GUI era. We just don't know exactly how that GUI is going to look/work. We can think and speculate, but keep in mind that as programmers we have an historical bias towards text-based solutions.

Why always about syntax??

Maarten Hazewinkel



Deus 12 posts since

Jan 3, 2005 2. Re: **Why always about syntax??** Jan 7, 2005 10:50 AM

I agree, and I mentioned to leave text representation as an option, with all stuff we use today (text refactoring, etc.), but I just seem that that particular period, between cell-based LOP and GUI-based LOP could be its pitfall.

I mean, I personally like the idea of developing new languages. Heck, I got used to idea of developing classes when I was working only with function/structure programming types, but after all, I am only 23, and in my firm there are people which aren't over 45 and constantly find arguments to battle against OOP.

I want to say, somehow I think that bending from structure/functions to classes wasn't SUCH a big step - there is still a problem and number of techniques and tricks to implement solution - but in LOP WE should be developing those techniques and tricks. Its somewhat like regular power-drill - when someone invented it, everyone can use it without much problem; the invention is the biggest one.

So there will be courses, tutorials, mind-shapening guides but in the end, I think 90% of programmers who work long time and have skill in OOP will think something like "So I have to exchange something which works good for me to something I know completely nothing about, and will take me few years to get accustomed to it?". Sure, there are people like on this forum, who like to explore new ways and stuff, but most of us (homo-sapiens) "have tendency to attach and hold current state at any cost" (said Siddhartha).

When you say that more specialised editors stay further form the abstract syntax tree; if those editors were developed in DSLs itself, I don't see a problem, if DSLs we developed editor in work just fine?

Why always about syntax??

When first reading about LOP, I imagined one of many places where I have to write ARGB color value, in form of 0x00A0B0C0 or some similar color, and if I could click that cell and select "ColorCheckerEditor" and select a color in some palette (maybe WindowsCommonDialogs), and in compile-time, that color would be translated to hex-value.

For the function, I could make - for the sake of the argument - a smart code generator. If I connect those two, I could make that function at any time in two single clicks.

If I apply this to every possible statement in my code, I would end up in clicking and no typing at all. If I find things that are common between different statements, I would improve editor and reduce clicking and refactoring. By the time, editor would be specialised, and at any time, it could represent text, and still would be near syntax tree.

Am I wrong somewhere in my vision of LOP, or just too optimistic?



[Rob Harwood](#) 896 posts since

Sep 10, 2002 3. Re: **Why always about syntax??** Jan 9, 2005 3:03 AM

The only danger of optimism is disappointment. 😊 If you can handle disappointment, and be realistic about the success of new ideas, then optimism is a great thing!

Regarding advanced GUIs/editors: We will be developing at least one DSL dedicated to arbitrary GUIs. One idea floating around is a language to represent Model View Controller. We'll be looking closely at Smalltalk, that's for sure!

From your comments, I think you would enjoy reading Thomas Kuhn's 'The Structure of Scientific Revolutions'. OOP **was** a big change when it came about. For you, and for me, it was natural. Read Kuhn to understand why.

Regarding typing vs. mouse. I don't think the keyboard is in any danger of extinction! 😊 There are many things which the keyboard makes much easier, especially for expert users,

Why always about syntax??

which programmers certainly will be, for any software tool. Clicking is good, but many times the keyboard is much faster. Ctrl-Space!!!



[Rob Harwood](#) 896 posts since

Sep 10, 2002 4. Re: **Why always about syntax??** Jan 9, 2005 2:52 AM

You are absolutely correct on all the issues, Maarten.

One thing I would like to point out is that 'cell-based' editors can be **very** flexible. Most form-style interfaces (Swing panels, standard Swing controls) can be emulated by using a cell-based editor. If you consider 'configuration' as a special language, for example, IDEA's various configuration settings dialogs, all of these could be emulated with cell-based editors. This opens up a wide range of everyday interfaces that can already be handled by cell-based editors. Advantage: The editor is 'aware' of what you're doing and can help you with various features such as code-completion, etc. Something to think about.



[Hendy Irawan](#) 8 posts since

Jun 16, 2009 5. Re: **Extending MPS "editor language" to support graphical visualization & editing** Jun 17, 2009 8:03 PM

Deus, Maarten, and Rob, I agree with you guys in your unique respective views.

Cell-based editors are very flexible, and right now I think it's easy to write many things/models in MPS' "text-based" structured editor.

However a graphical editor/visualization is desirable or even necessary in some cases. And I think it's possible by "simply" (it won't be simple) using an extended editor language and implementing it.

As the editor right now is visualized based on AST, and the editor for a Concept itself is described using Editor language, then by:

1. extending the Editor language with new concepts, for example "RadialLayout", "Glyph", "Circle", "Connector" etc.

Why always about syntax??

2. implementing (visualization, editing, or both) support for those new concepts in the editor implementation

it should be possible to do graphical editing.

Since the exact same editor is used to edit ConceptDeclaration, Concept Editor, and a model (Concept instance), JetBrains could've provided a "graphical editor" to edit a Concept or a Concept Editor or the other things for us to use. And since we're using the extended editor language to design the editor for our Concept, that means we can use graphical structures in it and therefore, our Models become graphically editable. :-)

Please tell me that the above scenario is possible/planned ?

Now, when that happens, I sure will want to extract the MPS viewer/editor somewhere else outside MPS. :-D *grin*