Proposals for the Reborn Pharo Developer

Accepted to IWST09

Simon Denier Damien Pollet Stéphane Ducasse

RMoD Project-Team, INRIA Lille-Nord Europe

{simon.denier, damien.pollet, stephane.ducasse}@inria.fr

Abstract

Smalltalk was at the birth of current IDEs. Current Smalltalk IDEs, however, lost their abilities to adapt to developer needs (edit and jump, back button, auto-completion,...). Therefore while offering a powerful sets of tools current Smalltalk IDEs looks clunky and often lacks the application of a consistent set of guidelines. In this paper we sketch some possible IDEs future features or reorganization.

General Terms IDEs, User interfaces, code browser

1. Motivations

Pharo wants to vivify the Smalltalk experience. Hence, the development experience with Pharo needs to be vivified.

Although Smalltalk has supported from the beginning the use of visual interfaces to browse and program systems, it now suffers from the age of its visual tools, which have basically not changed since their inception. This appears in particular in Pharo, in which development tools have seen few improvements.

We think that the Pharo experience needs to be enhanced not only with new tools, but also with new guidelines and principles around which the development should be centered. In nowadays systems where methods, classes, and packages grow everyday, we think that two capabilities are of primal importance for the developer: the ability to focus on a small set of relevant items and the ability to navigate and discover the system at different scopes and through different views.

2. Guidelines for Pharo Tools

Guidelines are simple keywords which drive the specifications and design of Pharo tools. This set allows one to assess which guideline a tool targets in priority and how well it fares with the others.

focus the developer only needs a small subset of the system at one time;

context context is tailored to not disturb the developer yet provide opportunities to expand the focus;

feedback the system should provide proper feedback in space and time;

ubiquity seamless interaction everywhere;

discoverability the developer can easily discover new aspects of code;

incrementality support incremental development;

consistency and efficiency system interaction is consistent and efficient.

3. Existing Tools for Development

Seeveral attemps have been performed to enhance the default set of tools. Here we give a non-exhaustive list with some key comments.

- Package Browser: it integrates the notion of packages and provides some start at supporting better navigation. Smart groups are provided for focus but deserve a deeper integration. A good work has been done to support traits and inheritance/package navigation. Emergence of important items been placed to the top of a list is interesting because they help developer focus by distinguishing changes.
- Enhanced Explorer/Inspector [Plua]: it merges and improves two tools, the inspector and the explorer. Basic Inspector should only be used per se in the debugger, otherwise this better version should be invoked.
- Chasing browser: the basic implementation lacks visual support to show the navigation context (*"what is displayed in one pane?"*) and to follow the flow of sliding panes, failing feedback. An extended version with smooth animation solves the problem of navigation feedback with the sliding panes [Plub].

IWST'09 2009/08/??, Brest.

- Whisker [Way]: it offered a really efficient way to display and contextualize inside a single browser multiple code panes, showing different methods from one or multiple classes. Whisker is currently unmaintained.
- Starbrowser [Wuy]: it proposed an extensible browser based on smart groups, leveraging focus for the developer.
- OmniBrowser [BDPW07]: it is an extensible framework for browsers, which represents navigation and pluggable components in a metagraph. However, flow of control is often limited. It should be noted that the OmniBrowser framework has targeted the development of such new tools, but did not try to reinvent the browser, rather offering a new implementation of the same principles which was easier to enhance.
- Browser Booster [Rob]: it is a package that supports better navigation using double-click, history navigation and the possibility to have multiple unsaved code panes in the browser.

Other development tools, such as Shout, eCompletion, or OCompletion, leverages the developer experience. However, they are more concerned with the syntax and semantics of the language and we do not discuss them. Their tight integration with the graphical tools is a key element in their acceptance though.

4. Principles and Proposals

Principles define some general functions of the environment, such as code navigation or system interaction. Principles combine multiple guidelines to achieve good user experience. Each of the following subsections presents and discusses one principle, relates it to our core guidelines, and make new proposals to embody the principle in Pharo.

4.1 Focus of Attention

Focus of Attention is the principle which should require top priority in new development environment. The rationale for this principle is that development is task-driven and that any developer works on a small subset of system at one time. Even when browsing a system, the developer has to be able to come back to the few items which are at the center of his interest.

Focus of attention of course directly embodied the focus guideline but it also needs support from context, discoverability, and incrementality: the developer should be able to discover related items to expand its focus if necessary. Moreover, this can be done incrementally as the developer refines its task.

Working Set. We propose the concept of *working set* to enable this principle. Any item (class, or method) deemed of interest for the current task goes in the working set at the request of the developer. Such pinned items stay in the working set until intentionally removed.

The working set can be enhanced with a degree-of-interest model to access recent items in navigation history. Indeed, items recently browsed and moreover recently edited imply some interest of the developer. Contrary to pinned items, they may disappear automatically when interest vanishes. To keep its efficiency, the working set must stay reasonably small and may be automatically cleaned after a while (for example of recent history items).

Dedicated Code Browser. The code browser also needs to be rethought around the working set to enable full focus for the developer. We propose to evolve the current browser design in a dedicated working set browser so that browsing panes at top do not show the whole system but only the working set. Showing the working set at any time provides better context and interaction, as one should be able to switch between any items of interest in as few clicks as possible.

4.2 Incremental Refinement

Incremental Refinement should enable and support the incremental nature of nowadays development, especially in a dynamic environment such as Smalltalk.

Good incremental refinement guarantees that code tools can support, or at least not clog up, the cognitive process of the developer. Such a process implies multiple simultaneous modifications when the developer needs to work out some collaborations between items, or simply when fixing a related item while coding a method.

We propose that code browser should support multiple code panes in the same browser, to save space and enhance side-by-side relations of code. It should also provide a better support for **unsaved** code panes. Code panes can be left unsaved while browsing a related item, enabling the developer to quickly update or check its understanding of the system while refining the code. Unsaved panes have automatically top priority in the working set, as they point to the current items of interest.

4.3 Context and Feedback

Context and Feedback are general concerns of any UI. It states that the environment should provide the user with information about its current state and location in the system (class, method). Any reactions to user inputs must be done in a spatially and timely fashion, i.e. the UI must be responsive and display result where the user expect it or can take a glimpse at it. *Context* also complements *Focus* by providing the developer with opportunities to expand its focus.

Currently, some aspects of Pharo UI fail to provide proper context and feedback. For example, the display system in Pharo uses sophisticated algorithms to choose the location for new windows, based in general on the empty space. However, such algorithms makes it hard for the user to predict where new windows pop up and even harder to track windows popping up everywhere. Other tools fail to provide the

right information or the right feedback to tell the developer what is happening. This is for example the case of the basic chasing browser, where it is difficult to follow the flow of new panes and where the search panes can sometimes display heterogeneous information (such mixing method lists and method implementors).

This matter can only be addressed by many small choices in UI design, relevant to each tool. However, we propose to address the lack of predictability in new windows by making them appear with their title bar right under the mouse pointer, so that clicking will allow to drag and drop the window wherever the user wants it. New extension Object Finder [Plub] remediates the problem with smooth sliding, providing a visual clue for changing context.

4.4 Ubiquitous Interaction

Squeak and Pharo both manifest some *Ubiquitous Interaction* capabilities with the ability to interpret any text selection in the environment as small snippets of code which can be executed or browsed. This enables a seamless experience between the different tools of the environment, mixing code and objects —a feature rarely seen in other environments.

However, not all interaction modes are so seamless in Pharo. For example, menus are the main mode of interaction to perform a wide range of operations, such as opening a browser, saving the image, refactoring some codes. The classic drop-down menu is less accessible than a toolbar; it is cumbersome for most-used items as one easily bypassed an item in the menu list. Besides menu organization can become messy with lots of addition if left uncontrolled.

Ubiquitous interaction should be enforced for tools as it is for code. We also propose that multiple means of interaction be included. Key bindings must be consistent across the environment. A keyboard launcher such as Algernon [HH] complements bindings with discoverability.

Pie Menus can replace contextual menus for most used actions. A pie menu is usually divided into six to eight slices, each slice launching a command. It offers discoverability as one can easily see the available options, and consistency as each action can be memorized to be in a particular slice, speeding the lookup in the menu.

Hyperlink navigation also favors ubiquity as one follows its interest by clicking the items. The idea is that each language element in a code pane can become an hyperlink. Hyperlink navigation should be semi-modal, i.e. only happens when the user presses a combination of keys. For example, `Control+click` can browse implementors and definitions while `Control+Alt+click` can browse senders and references.

4.5 Focusable Navigation

Current navigation in Pharo is as powerful as it is ubiquitous. Any text snippet anywhere can be browsed if it happens to be a class, looked up for implementors or senders for message. However, such search are performed system-wide, lacking the focus the developer often needs. In a few cases, some scoped requests are defined such as *hierarchy senders* but even this request performs a system-wide search before filtering results with the scope.

Refactoring Environment circumvents the problem by building almost arbitrary set of classes and methods, restricting search and refactoring to the selected set. However, it lacks the ubiquity of the standard framework as it is cumbersome to launch from contextual menu.

Focusable Navigation mixes focus, context, and discoverability to allow the developer to discover new aspects of the system yet restrict the information retrieved to only pieces which are explicitly in the scope required by the developer.

Navigation Framework. We propose a new search/navigation framework where any request can be expressed with the following principle: *Look for target with aspect in scope*. Three parameters define the request:

Target the text snippet (selected item in pane, under mouse, text selection) which the user wants to look for (for example, a class name, a message, any string);

Aspect the kind of lookup to be performed, for example definition of the target (class definition, method implementors) or dependencies to the target (class references, method senders, variable accessors);

Scope scope of the lookup, restricting the search space.

This framework postulates that *"looking for String definition in the system"* is not conceptually different than *"looking for #add: senders in the package enclosing OrderedCollection"* or even *"looking for Collection methods in hierarchy of Collection (including inherited methods)"*. However, the implementation of each of those request may largely differ for optimization. The purpose of the framework is to provide a uniform interface to search and navigate in the system while providing focus.

A target is virtually any text selected by the developer and can be a message, package name, class name, variable name, symbol, string… Aspects come as a list of operators while scopes have their own range. Aspects can be:

definition	implementors, class definition, instance variable definition
dependencies	senders, references, variable accessors
readers	instance variable readers
writers	instance variable writers
container	items (class, package) containing the target
comment	method comment or class comment
message list	list of messages sent in method body, method dictionary of a class
substring	in a string of method body
source	part of method source

Scope ranges in decreasing order of magnitude from System to method scope:

system	all system
working set	packages enclosing items in working set
class hierarchy	can be refined in all hierarchy, only superclasses, or only subclasses
package	
merged class	instance-side and class-side
instance-side	only instance-side
class-side	only class-side
method	method body

Not all combinations of target, aspect, and scope are relevant. For example, looking for definition of a target which is neither a class, a method, or an instance variable should yield no result. Looking for writers to a target which is not an instance variable makes no sense.

Navigation Browser. To support such a framework, the UI must be adapted accordingly. We propose to create a new navigation browser merging the capabilities of the system browser and the chasing browser. Such a browser should allow to navigate seamlessly from packages to classes, methods, senders using toolbars to configure aspects and scopes at each level of exploration.

The navigation browser can be integrated with the code browser through working sets (Section 4.1). For example, double-clicking an item in the navigation browser would add the item to the current working set.

5. Conclusion

We have developed a set of guidelines to assess the quality of UI and tools in Pharo. From these guidelines we devise some principles which represent nowadays expectations for the developer experience in Pharo. We make some proposals to embody those principles, hoping they will spark some discussions or even some contributions.

Two proposals seem particularly important to our eyes. First, the capability of focusing work on a few items with working set in the code browser. Second, the capability to browse the code at different scope and through different aspects with the navigation framework. Both proposals should help the developer to manage the ever growing complexity of nowadays project.

References

[BDPW07] Alexandre Bergel, Stéphane Ducasse, Colin Putney, and Roel Wuyts. Meta-driven browsers. In *Advances in Smalltalk — Proceedings of 14th International Smalltalk Conference (ISC 2006)*, volume 4406 of *LNCS*, pages 134–156. Springer, 2007.

[HH] Joey Hagedorn and Erik Hinterbichler. Algernon. http://www.squeaksource.com/Algernon.html.

[Plua] Frédéric Pluquet. NewInspector. http://www.squeaksource.com/NewInspector.html.

[Plub] Frédéric Pluquet. Object Finder. http://www.squeaksource.com/ObjectFinder.html.

[Rob] Romain Robbes. Browser Booster. http://www.squeaksource.com/BrowserBooster.html.

[Way] Doug Way. Whisker: The O-O Stacking Browser. http://www.mindspring.com/ dway/smalltalk/whisker.html.

[Wuy] Roel Wuyts. Star Browser. http://wiki.squeak.org/squeak/2935.