An User Interaction Aware System Browser

Roberto Minelli
Benjamin Van Ryseghem

December 10, 2013

1 Introduction

A system browser is the main tool of a Smalltalk developer as it provides easy access to the whole system source code. From the early days of Smalltalk to today the system browser has proposed a workflow statically described by the system browser architects. Due to this static description, the way to use the system browser is not really adapted to the user needs. Even if some implementation of this tool (OmniBrowser or Nautilus by example) proposed some mechanisms to bend the system browser workflow to fit the user workflow. But so far this implementations were based on some extensions that the user, who is also a coder, can add by implementing new behaviours.

We would like to reimplement Nautilus (the current Pharo system browser) using Spec. This new framework provides the possibility to dynamically change the user interface. Such a feature combined with the ability to collect user interaction data provided by DFlow will be used to make the system browser as close as the user interactions as possible.

The goal is to simplify the usage of the system browser which is the back bone tool of the system. To achieve this goal we would like to have a new system browser that can dynamically change the way information are presented in order to ease its usage for each user.

2 The current system browser

The current system browser looks the same since the early ages of Smalltalk even if multiple implementation has existed. It is horizontally split in two part:

- the top most panel to navigate through the system;

- the bottom most panel where the source code is displayed and can be edited.

The top most panel is itself split in four columns representing the levels of Smalltalk entities. From the left hand colin to the right hand column there are:

- the packages list, including all the packages of the system;
- the classes list of the selected package;
- the protocols list of the current selected class;
- the methods list of the current selected protocol.

An example a such a layout can be found in the current Pharo system browser, Nautilus as shown on Figure 1.

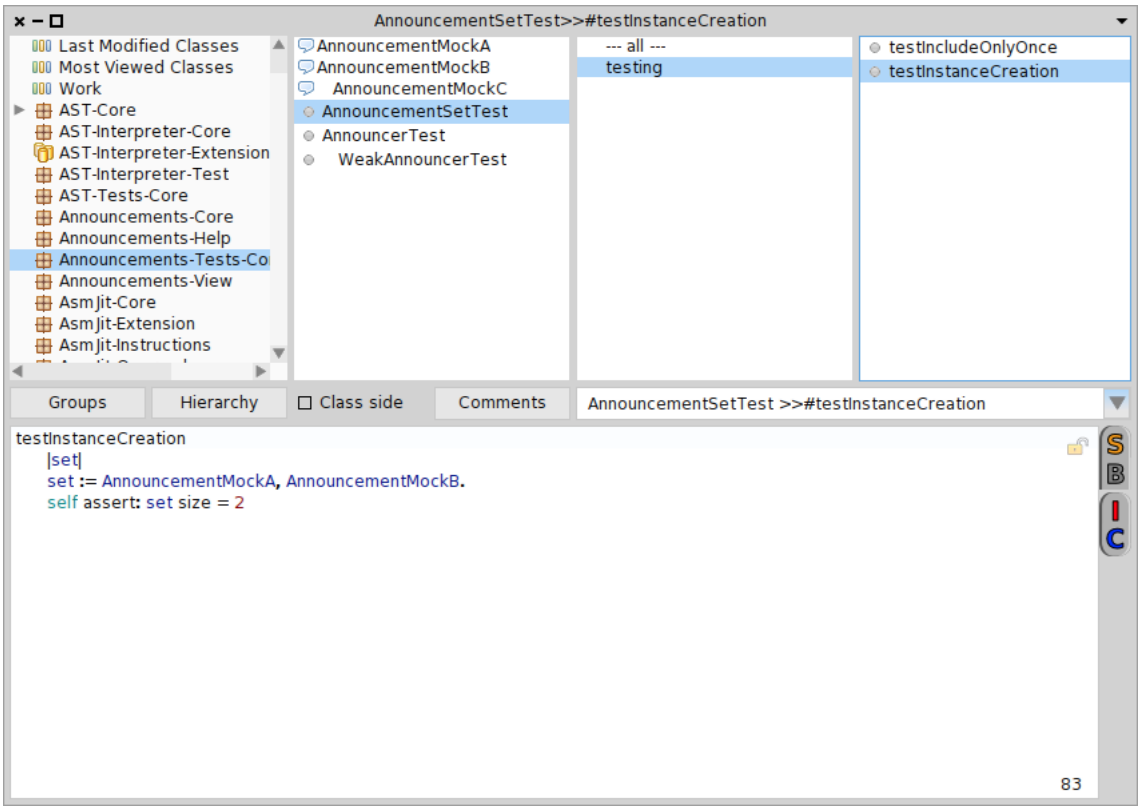

Figure 1: Pharo system browser: Nautilus

The layout itself of the system browser forces the users to accommodate to a strictly defined workflow. In the latests implementations of the system browser

(OmniBrowser or Nautilus) few mechanism has been introduced to try to tackle this lack of flexibility. But so far, those solutions (plugins, preferences) either require internal knowledge from the user or are still static in the sense that the changes are made once and for all.

Another limitation of the current implementation of the system browser is that its made fully using Morphic which only few Smalltalkers know. Due to this, it is complicated for a regular user to even change the simplest Nautilus behaviour.

3 Spec

Spec is a new framework in Pharo used to easily build and reuse widgets and applications. Contrary to Morphic, Spec is fully event based, providing an easier to understand workflow. It also provide a syntax that is closer to a natural event workflow, making coding application simplier.

In addition, Spec provides a dynamic way to build widgets which leads to the capability for the developer to change the widget layout or sub widgets on the fly. Even if this can be done in plain Morphic, it is often a tedious task where in Spec it is completely seamless for the user. Spec supports the addition and removal of sub-widgets on the fly providing user more flexibility.

Spec also propose a simple layout management with multiple convenient methods bridging the gap in Morphic where layouts are complex. The code sample Code 1 and the code sample Code 2 represent the creation of the similar widgets. If the result is similar it appeared that the Spec version is understood more readily than the Morphic equivalent.

```
MySpecExample>>#defaultSpec

^ SpecLayout composed
    newColumn: [ :column |
        column
            add: #top height: 26;
            add: #middle;
            add: #bottom height: 26
    ];
    yourself
```

Code 1: Spec widget layoutting

```
MyMorphicExample>>#buildWidget

^ PanelMorph new
    changeProportionalLayout;
    addMorph: self top
    fullFrame: (LayoutFrame identity
        bottomFraction: 0;
        bottomOffset: 26);
    addMorph: self middle
    fullFrame: (LayoutFrame identity
        topOffset: 26;
        bottomOffset: -26);
    addMorph: self top
    fullFrame: (LayoutFrame identity
        topFraction: 1;
        topOffset: -26);
    yourself
```

Code 2: Morphic widget layoutting

4 Challenges

Several challenges need to be faced in order to reach our goals.

The first challenge will be to filter the user interactions to a representative sample group. Among all the interactions made by the user only few are made often enough to be relevant. The interactions need to be grouped by categories representing a type of user interaction [1], allowing small variations within interactions of a same category. Then only the relevant categories with a big enough population will be activated for interacting with the system browser workflow.

The second challenge will be to update the system browser workflow depending of the chosen categories. To achieve this, an action to update the system browser workflow must be defined for each category. But in addition the actions must be able to be combined, or at least know how to be combine with another action. Also some actions could occur only if multiple interactions categories are activated.

The last challenge will be to validate the actions. Indeed it may be difficult to find a pertinent action for every user. Then only a statistic approach mixed with tests on a sample group can lead to a generic enough solution.

[1] like the use of the command 'Search class'

5 Goals

In order to help the user in his/her navigation through the system a first goal will be to remove the graphical element never used. Indeed, in Nautilus a lot of basing interactions are accessible via shortcuts. A user knowing and using a specific shortcut will never use the graphical representation of the action (a button, a menu entry etc). Because of this, plenty of spaces is wasted for never used widget. The goal is to detect never or rarely used activators and to remove them on the fly.

A second goal is to enhance the visibility of the currently use part of the browser. As seen in the section 2 the current system browser is split in two parts. The top most part dedicated to the navigation, the bottom most part used for edition. The user always shifting from one usage to the other. But when the user is editing code, the navigation is not needed and is taking space without providing any useful information. The goal is to enlarge the code pane while the user is editing and by contrast expand the navigating pane when the user is browsing the system.