Alternative UX Extensions and Their Trade-offs for Code Completion in Pharo

Mehdi **Elkolei**, Omar **AbedelKader,** Stéphane **Ducasse**

Overview

1. Code Completion
2. Code Completion in Pharo
3. Code Completion Limitations in Pharo
4. Methodology
5. Implementation in Pharo
6. Conclusion

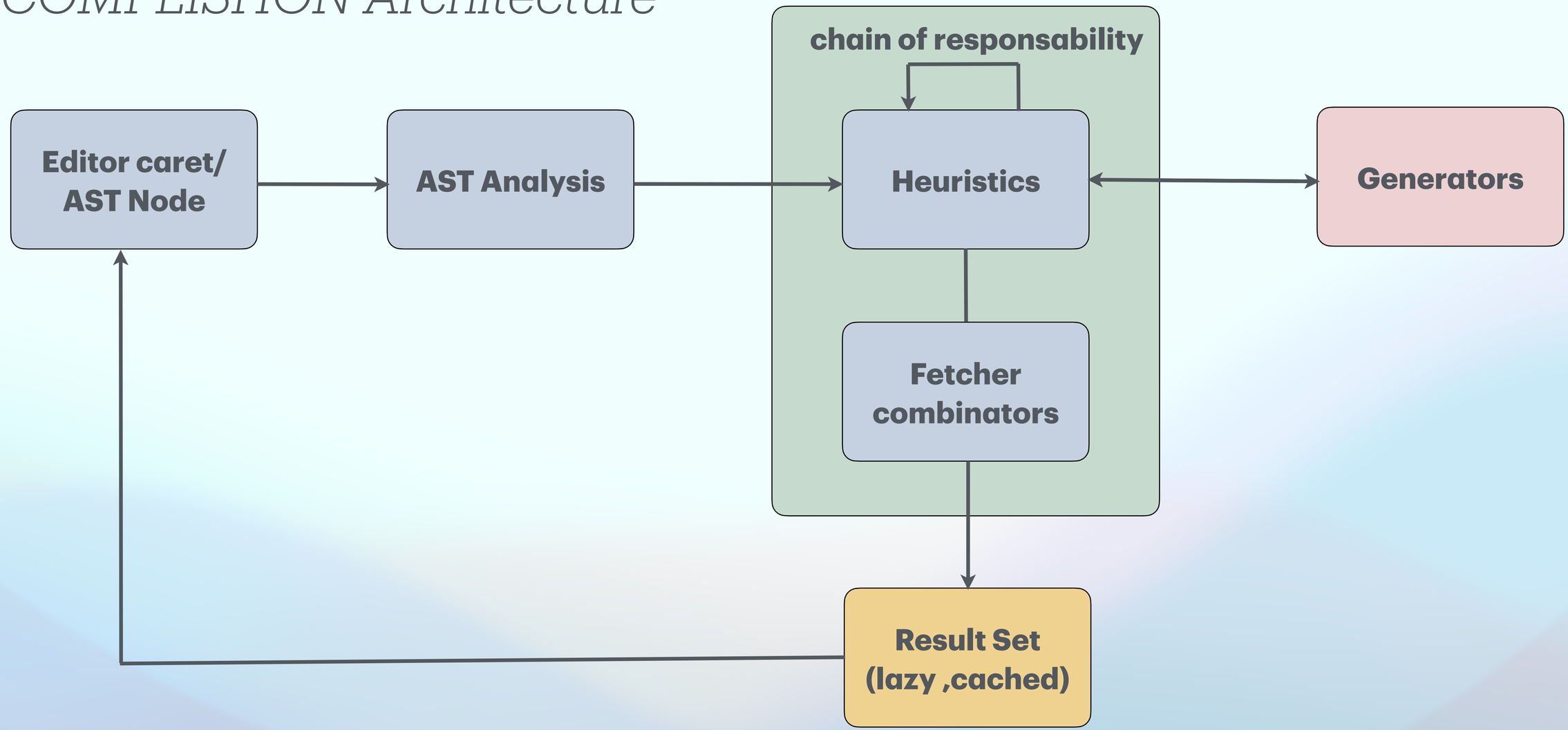
Introduction

Code Completion

- Predict the word or phrase the user is likely to type.
- Limit the amount of information entered by the user.
- Suggest relevant methods, classes, and variables.
- Improve productivity and coding efficiency.

Introduction

COMPLISHON Architecture



Limitations of Code Completion in *Pharo*

- COMPLISHON relies on exact prefix matching, making it highly sensitive to typos.
- Pharo's naming conventions force users to repeat contextually inferable prefixes.
- Relevant candidates can be overlooked in long completion lists.
- Pharo does not support camel-case abbreviation matching

Methodology

COMPLISHON Extensions

- Typo tolerance
- Implicit Prefix Expansion
- Grouping Entries
- Camel-Case Matching

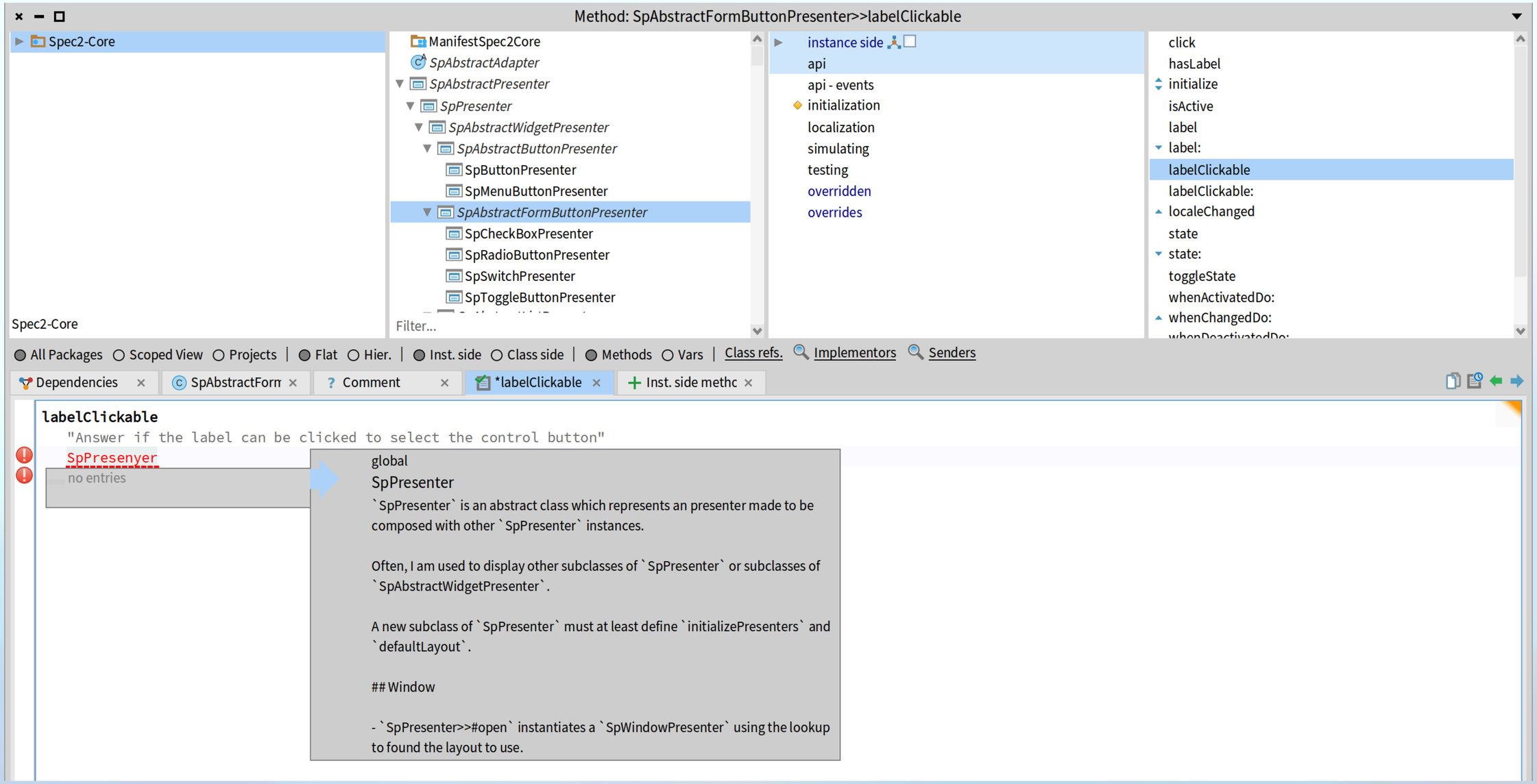
Extension I: Typo Tolerance

Limitations

Pharo depends on Strict Prefix Matching meaning :

- It relies on strict prefix matching between the typed input and candidate names.
- It is unforgiving of typos and small spelling errors.
- It depends heavily on the user knowing the exact name of the element.
- It prioritizes accuracy over flexibility.

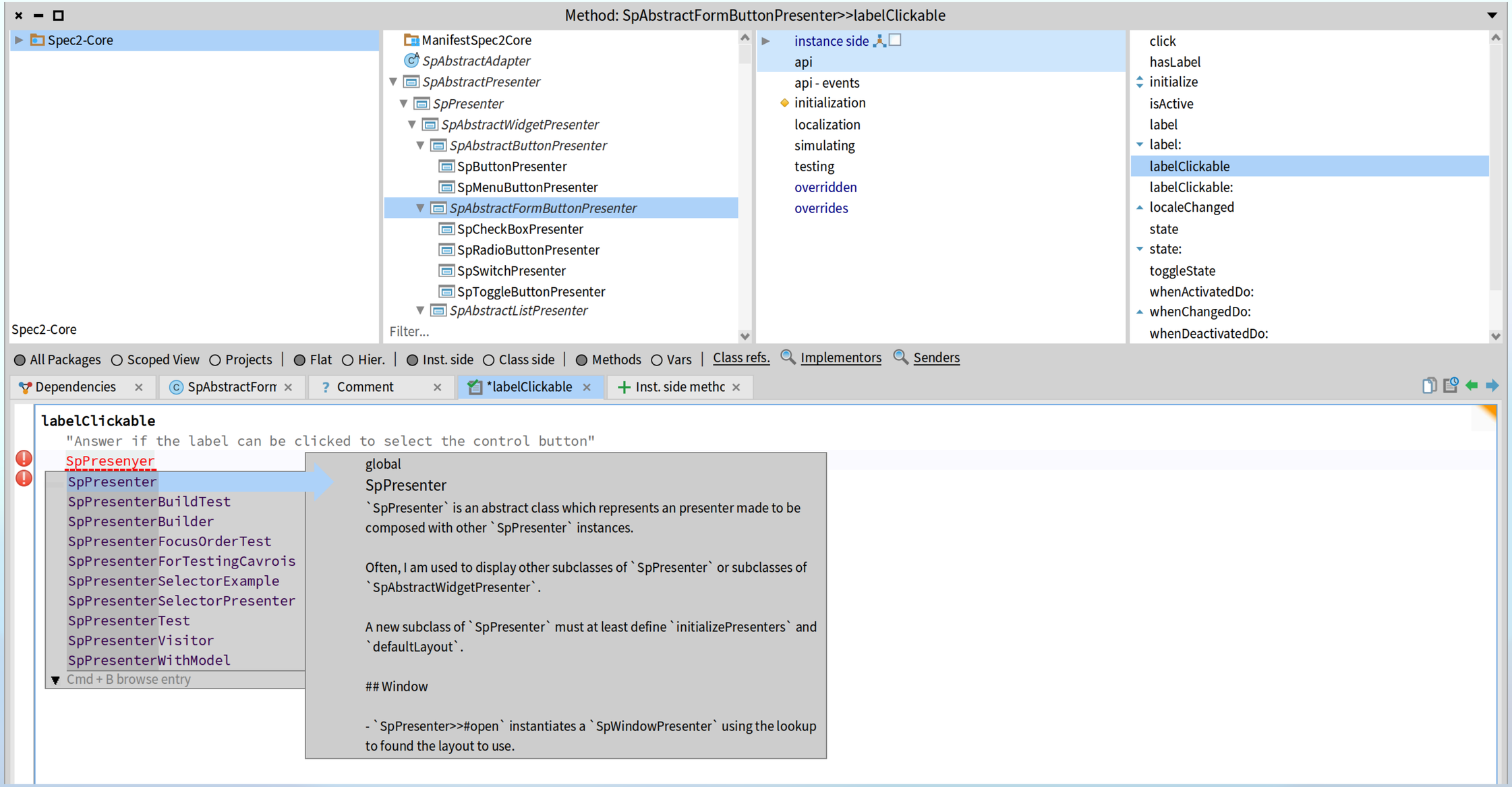
Limitations



Implementation

- Configurable tolerance: The main parameters are a tolerance rate in [0, 1] and a minimum token size.
- Edit-distance matching: based on Damerau–Levenshtein distance.
- Guardrails to reduce noise: To preserve relevant candidates while improving performance and avoiding noisy matches.
- Stable result ordering: To prioritize exact matches over fuzzy ones.

Impact


Method: SpAbstractFormButtonPresenter>>labelClickable
Spec2-Core
ManifestSpec2Core
SpAbstractAdapter
SpAbstractPresenter
SpPresenter
SpAbstractWidgetPresenter
SpAbstractButtonPresenter
SpButtonPresenter
SpMenuButtonPresenter
SpAbstractFormButtonPresenter
SpCheckBoxPresenter
SpRadioButtonPresenter
SpSwitchPresenter
SpToggleButtonPresenter
SpAbstractListPresenter
Filter...
instance side
api
api - events
initialization
localization
simulating
testing
overridden
overrides
click
hasLabel
initialize
isActive
label
label:
labelClickable
labelClickable:
localeChanged
state
state:
toggleState
whenActivatedDo:
whenChangedDo:
whenDeactivatedDo:
All Packages
Scoped View
Projects
Flat
Hier.
Inst. side
Class side
Methods
Vars
Class refs.
Implementors
Senders
Dependencies
SpAbstractForm
Comment
*labelClickable
Inst. side methc
labelClickable
"Answer if the label can be clicked to select the control button"
SpPresenyer
SpPresenter
SpPresenterBuildTest
SpPresenterBuilder
SpPresenterFocusOrderTest
SpPresenterForTestingCavrois
SpPresenterSelectorExample
SpPresenterSelectorPresenter
SpPresenterTest
SpPresenterVisitor
SpPresenterWithModel
Cmd + B browse entry
global
SpPresenter
`SpPresenter` is an abstract class which represents an presenter made to be composed with other `SpPresenter` instances.
Often, I am used to display other subclasses of `SpPresenter` or subclasses of `SpAbstractWidgetPresenter`.
A new subclass of `SpPresenter` must at least define `initializePresenters` and `defaultLayout`.
Window
- `SpPresenter>>#open` instantiates a `SpWindowPresenter` using the lookup to found the layout to use.

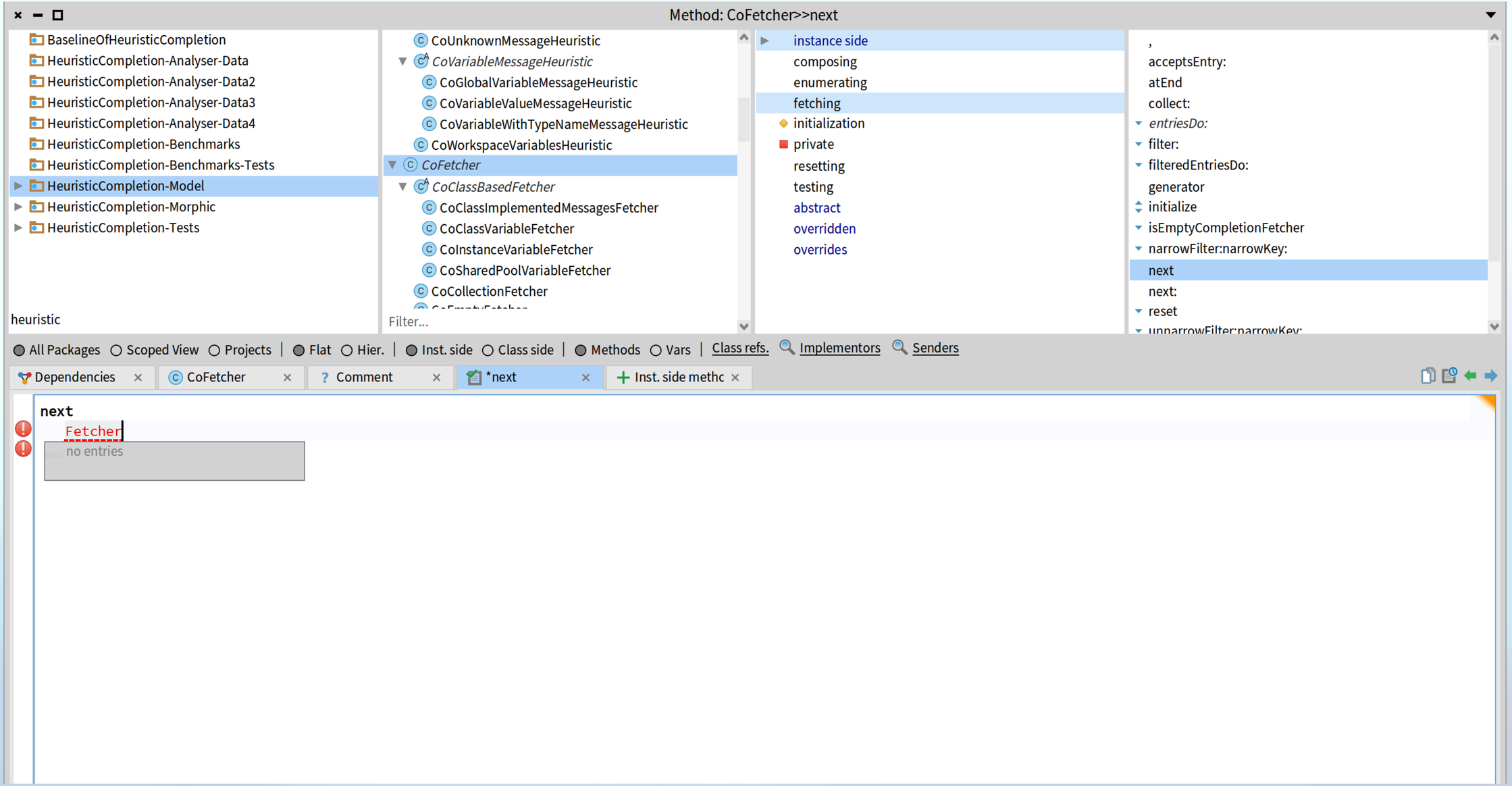
Extension II:
Implicit Prefix Expansion

Limitations

Pharo depends on naming conventions :

- Classes that are functionally related share a common prefix.
- The prefix is redundant when the context already implies the framework.
- Omitting the prefix should be enough to still get relevant suggestions.

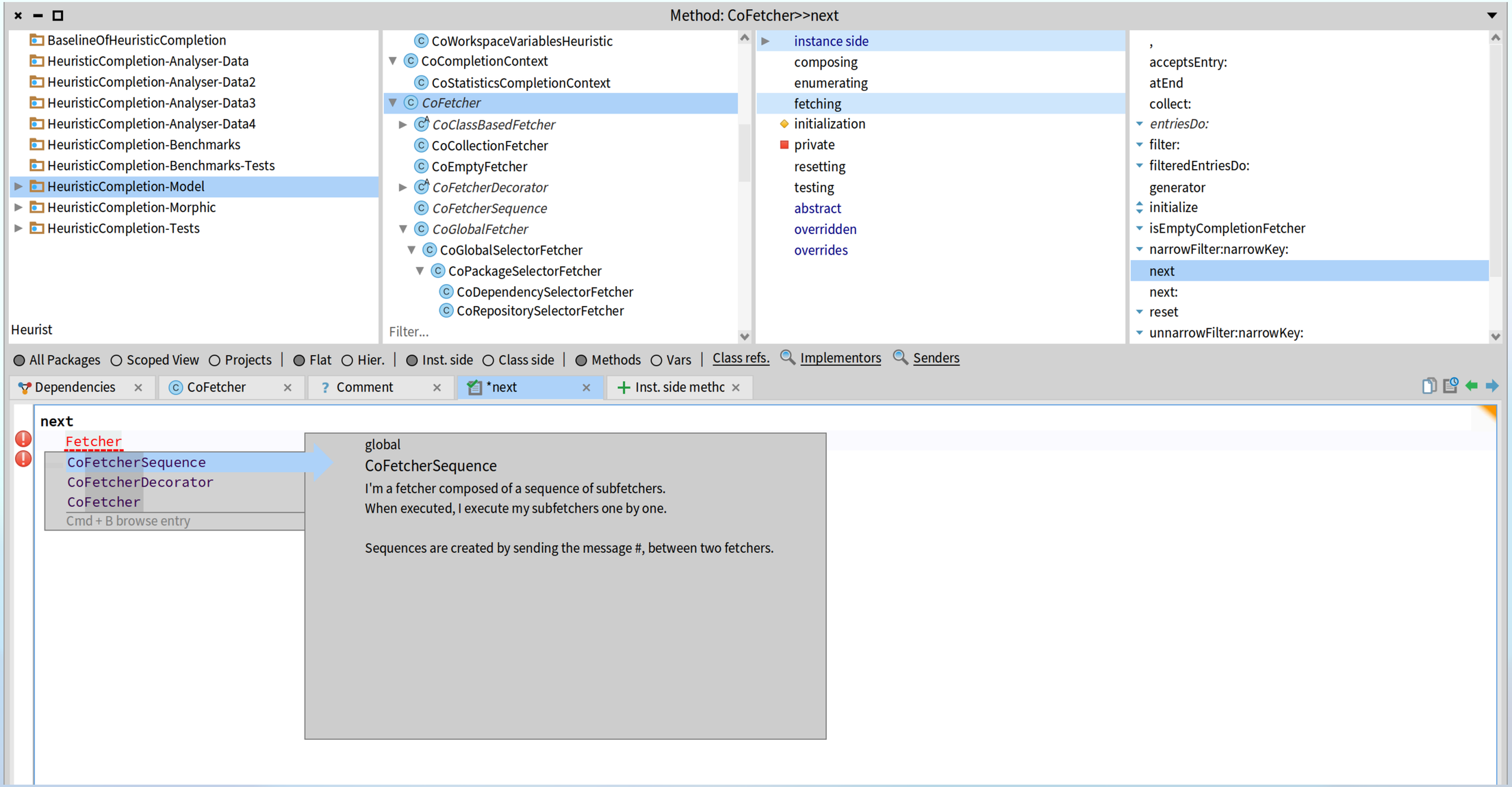
Limitations



Implementation

- Separation between matching and insertion
- Context-sensitive reduction of redundant typing
- Preservation of source-level explicitness:

Impact


Method: CoFetcher>>next
BaselineOfHeuristicCompletion
HeuristicCompletion-Analyser-Data
HeuristicCompletion-Analyser-Data2
HeuristicCompletion-Analyser-Data3
HeuristicCompletion-Analyser-Data4
HeuristicCompletion-Benchmarks
HeuristicCompletion-Benchmarks-Tests
HeuristicCompletion-Model
HeuristicCompletion-Morphic
HeuristicCompletion-Tests
Heurist
CoWorkspaceVariablesHeuristic
CoCompletionContext
CoStatisticsCompletionContext
CoFetcher
CoClassBasedFetcher
CoCollectionFetcher
CoEmptyFetcher
CoFetcherDecorator
CoFetcherSequence
CoGlobalFetcher
CoGlobalSelectorFetcher
CoPackageSelectorFetcher
CoDependencySelectorFetcher
CoRepositorySelectorFetcher
Filter...
instance side
composing
enumerating
fetching
initialization
private
resetting
testing
abstract
overridden
overrides
,
acceptsEntry:
atEnd
collect:
entriesDo:
filter:
filteredEntriesDo:
generator
initialize
isEmptyCompletionFetcher
narrowFilter:narrowKey:
next
next:
reset
unnarrowFilter:narrowKey:
All Packages
Scoped View
Projects
Flat
Hier.
Inst. side
Class side
Methods
Vars
Class refs.
Implementors
Senders
Dependencies
CoFetcher
Comment
*next
Inst. side methc
next
Fetcher
CoFetcherSequence
CoFetcherDecorator
CoFetcher
Cmd + B browse entry
global
CoFetcherSequence
I'm a fetcher composed of a sequence of subfetchers.
When executed, I execute my subfetchers one by one.
Sequences are created by sending the message #, between two fetchers.

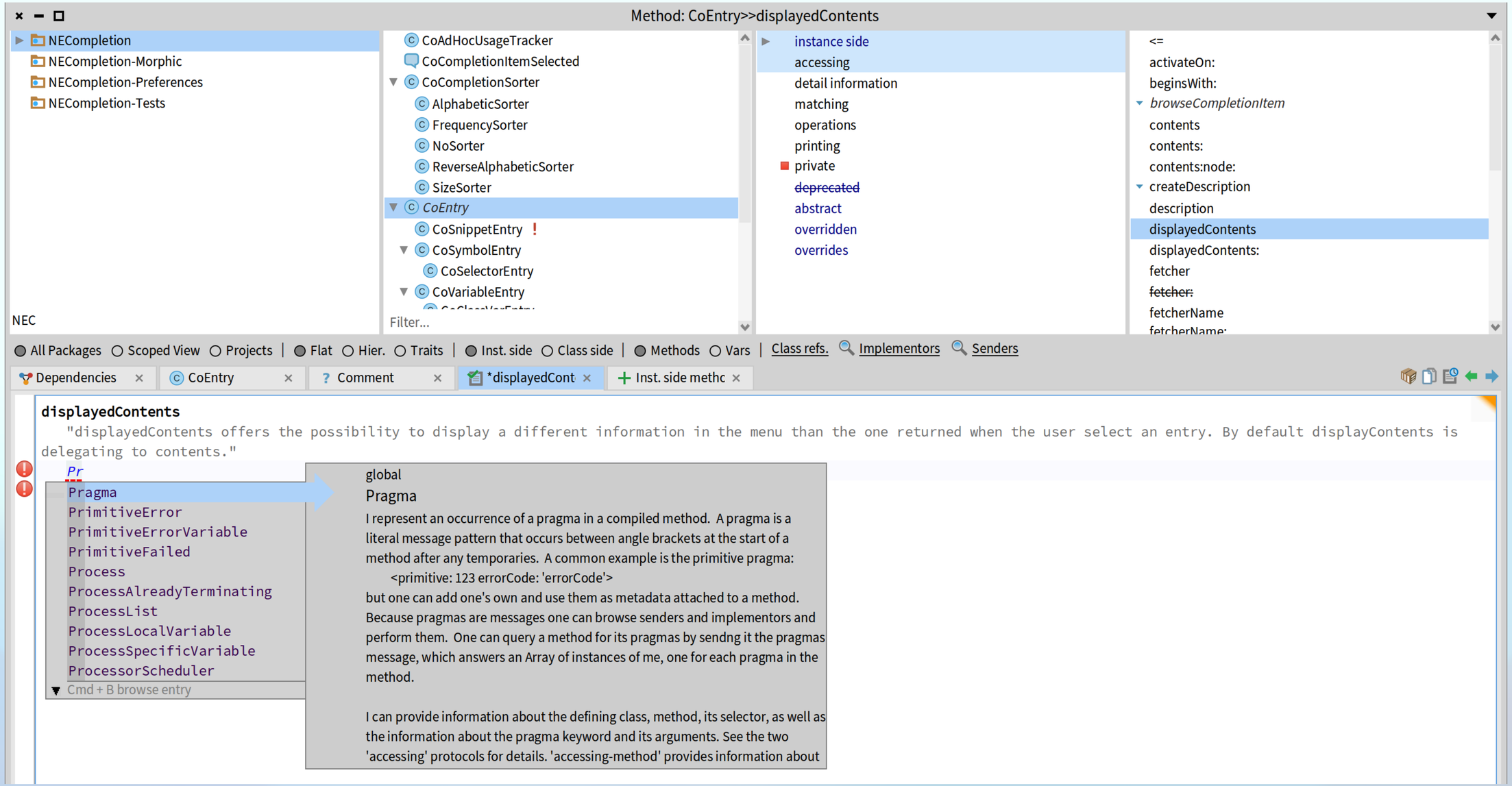
Extension III: Grouping Entries

Limitations

Once the candidates are calculated we might suffer from :

- completion lists show long runs of visually similar entries.
- Users must scan past repeated, uninformative prefixes to reach the parts that actually distinguish candidates.
- The issue is not with the algorithm itself, but with how it is presented.

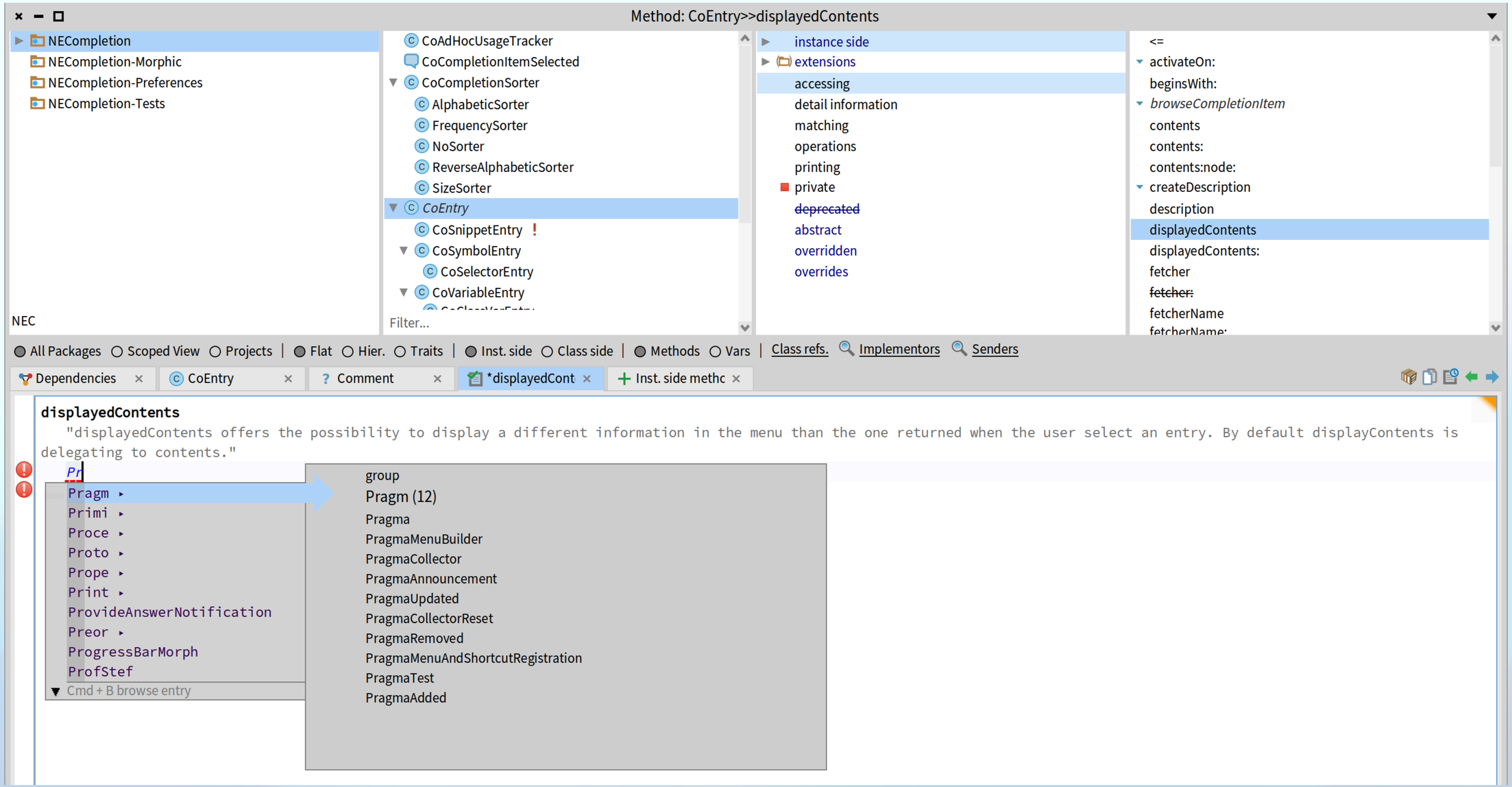
Limitations



Implementation

- Presentation should be separated from retrieval
- Grouping should reflect the structure already visible to the user
- Additional structure should remain lightweight

Impact


Method: CoEntry>>displayedContents
NECompletion
NECompletion-Morphic
NECompletion-Preferences
NECompletion-Tests
NEC
CoAdHocUsageTracker
CoCompletionItemSelected
CoCompletionSorter
AlphabeticSorter
FrequencySorter
NoSorter
ReverseAlphabeticSorter
SizeSorter
CoEntry
CoSnippetEntry
CoSymbolEntry
CoSelectorEntry
CoVariableEntry
Filter...
instance side
extensions
accessing
detail information
matching
operations
printing
private
deprecated
abstract
overridden
overrides
<=
activateOn:
beginsWith:
browseCompletionItem
contents
contents:
contents:node:
createDescription
description
displayedContents
displayedContents:
fetcher
fetcher:
fetcherName
All Packages
Scoped View
Projects
Flat
Hier.
Traits
Inst. side
Class side
Methods
Vars
Class refs.
Implementors
Senders
Dependencies
CoEntry
Comment
*displayedCont
Inst. side methc
displayedContents
"displayedContents offers the possibility to display a different information in the menu than the one returned when the user select an entry. By default displayContents is delegating to contents."
Pr
Pragm
Primi
Proce
Proto
Prope
Print
ProvideAnswerNotification
Preor
ProgressBarMorph
ProfStef
Cmd + B browse entry
group
Pragm (12)
Pragma
PragmaMenuBuilder
PragmaCollector
PragmaAnnouncement
PragmaUpdated
PragmaCollectorReset
PragmaRemoved
PragmaMenuAndShortcutRegistration
PragmaTest
PragmaAdded

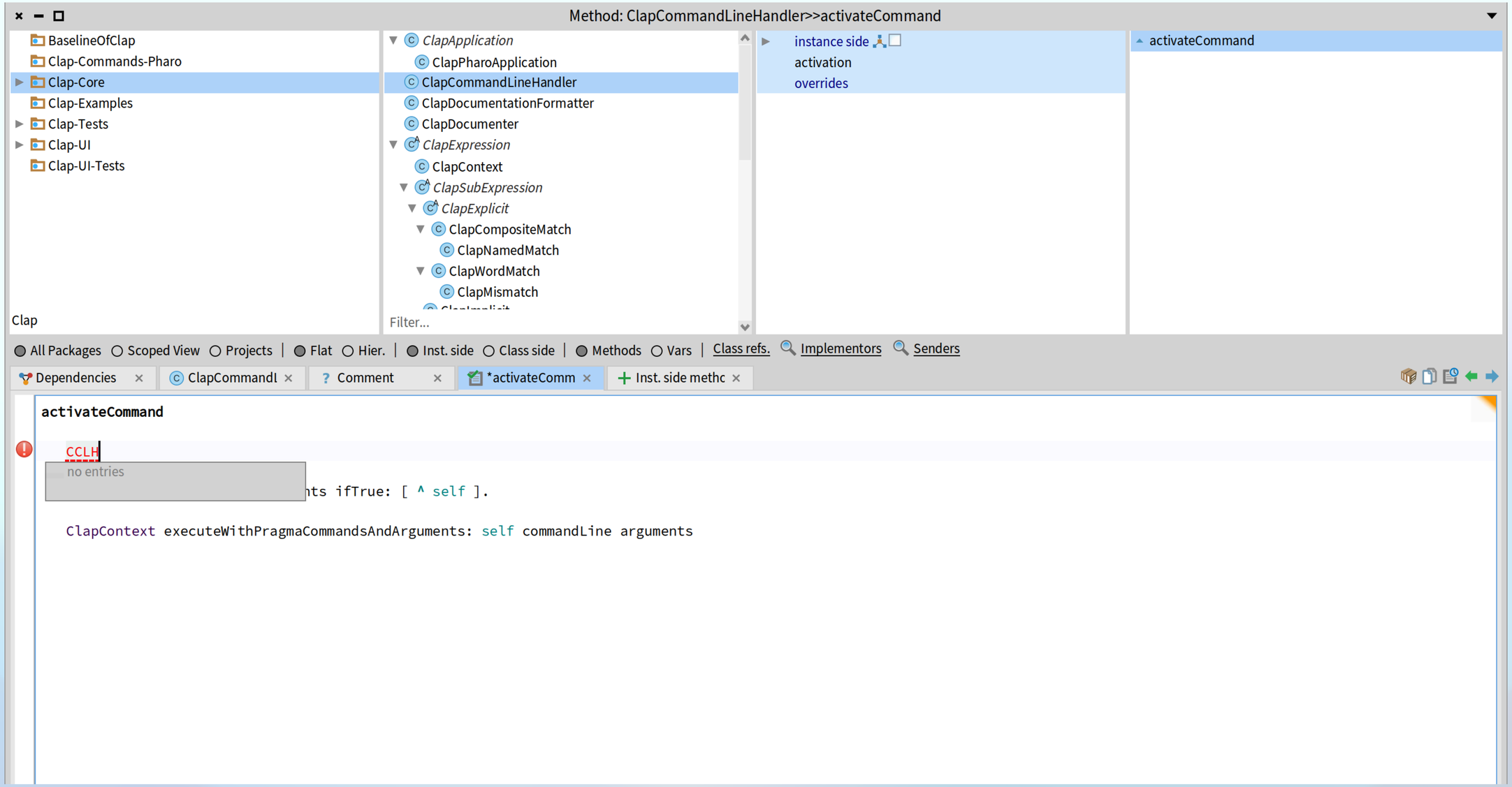
Extension IV: Camel-Case Matching

Limitations

Pharo is a purely object-oriented programming language :

- Pharo identifiers are long and descriptive by design.
- Camel case boundaries carry structural meaning that plain prefix matching ignores entirely.
- Typing only the capitalized characters of an identifier means nothing to a prefix-based engine.

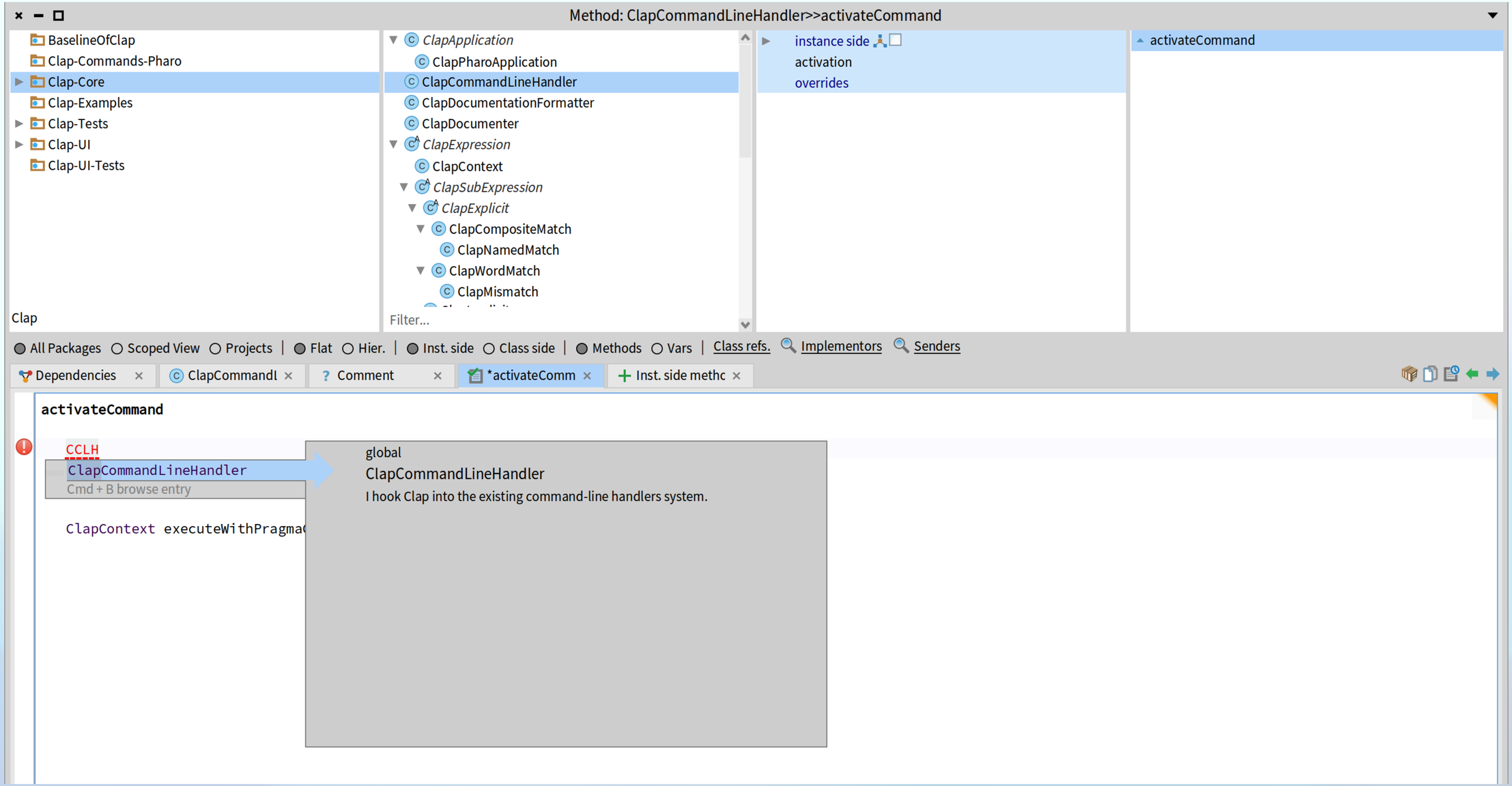
Limitations



Implementation

- Exploiting identifier structure
- Controlled flexibility
- Modular extensibility

Impact


Method: ClapCommandLineHandler>>activateCommand
BaselineOfClap
Clap-Commands-Pharo
Clap-Core
Clap-Examples
Clap-Tests
Clap-UI
Clap-UI-Tests
Clap
ClapApplication
ClapPharoApplication
ClapCommandLineHandler
ClapDocumentationFormatter
ClapDocumenter
ClapExpression
ClapContext
ClapSubExpression
ClapExplicit
ClapCompositeMatch
ClapNamedMatch
ClapWordMatch
ClapMismatch
Filter...
instance side
activation
overrides
activateCommand
All Packages
Scoped View
Projects
Flat
Hier.
Inst. side
Class side
Methods
Vars
Class refs.
Implementors
Senders
Dependencies
ClapCommandl
Comment
*activateComm
Inst. side methc
activateCommand
CCLH
ClapCommandLineHandler
Cmd + B browse entry
ClapContext executeWithPragma
global
ClapCommandLineHandler
I hook Clap into the existing command-line handlers system.

Implementation in *Pharo*

Implementation in *Pharo*

	Jaro-Winkler	Damerau Levenshtein
Output	Similarity Score	Edit Distance
Prefix Detection	Bonus for common prefix	All positions are treated uniformly
Complexity	O(n)	O(n·m)
Supported Operations	Matches & transpositions	Insertions, deletions, substitutions, transpositions

Conclusion

Conclusion

- Typo tolerance : more robust completion
- Implicit prefix expansion : less redundant typing
- Grouping Entries : improved readability of results
- Camel-case matching : better abbreviation support

Plugin are Available

- https://github.com/pharo-completion/typo-tolerance
- https://github.com/pharo-completion/implicit-prefix-matching
- https://github.com/pharo-completion/group-entries
- https://github.com/pharo-completion/camel-case-matching