

Interleaving of Modification and Use in Dataflow-driven Tool Development

Marcel Taeumel

Software Architecture Group
UCT-HPI Research School Workshop
April 14, 2014

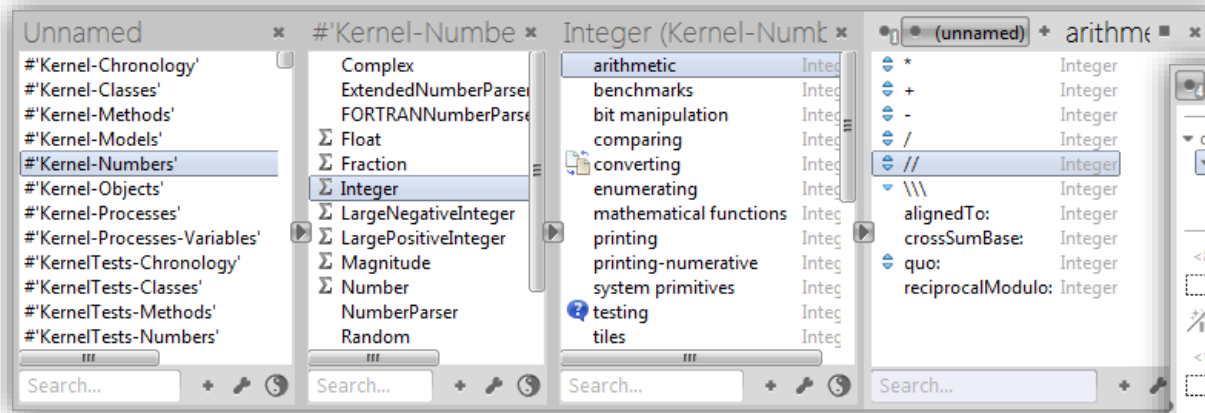
A New Programming Environment Prototype in Squeak/Smalltalk

2

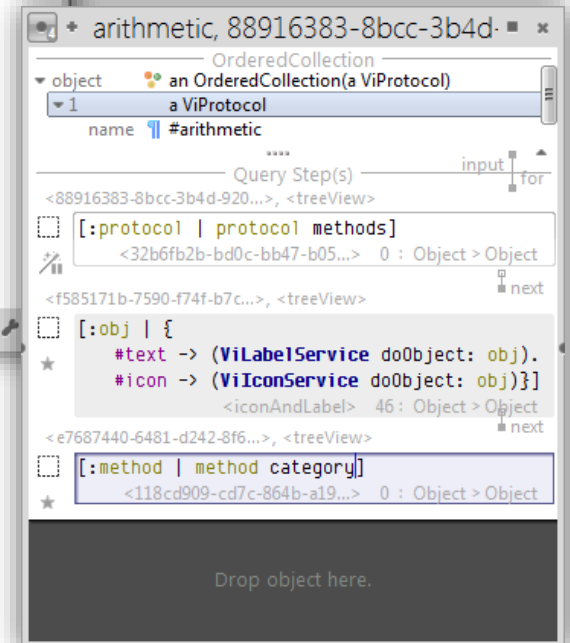
```
ActiveVivide openQueries: {  
  [SystemOrganization categories] asQuery.  
  [:category | SystemOrganizer default  
    classesInCategory: cat] asQuery.  
  [:class | ViProtocol protocolsForClass: class] asQuery.  
  [:protocol | protocol methods] asQuery.  
}.
```

OO Dataflow Programming

Live Adaptation



Convenient Tool Building

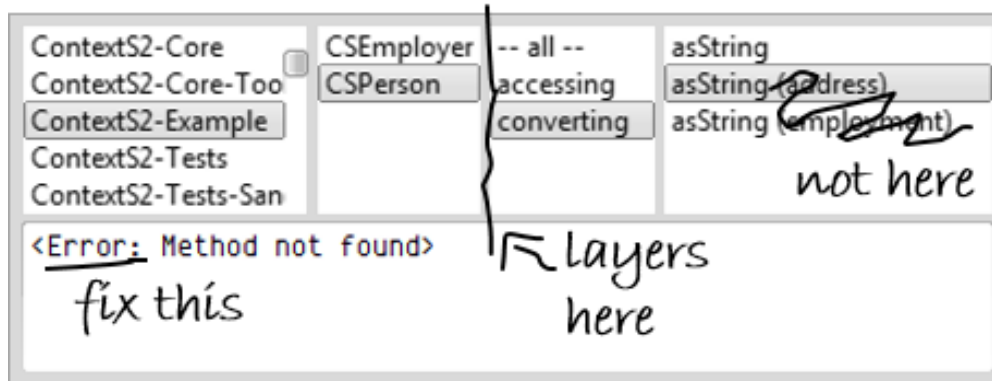


Iterative Tool Building

3

- **Exploration tools** support program comprehension tasks
- Programmers are capable of **building** such tools
- Tools in use entail **visual impressions** that trigger change requests

Example changes



The screenshot shows an IDE interface with a search results table. The table has columns for context, class, search term, and results. The search term is 'asString'. The results show 'asString(address)' and 'asString(employment)'. Handwritten annotations include 'not here' under the second result, a bracket on the left side labeled 'layers here', and '<u>Error: Method not found</u>' with 'fix this' written below it.

ContextS2-Core	CSEmployer	-- all --	asString
ContextS2-Core-Too	CSPerson	accessing	asString(address)
ContextS2-Example		converting	asString(employment)
ContextS2-Tests			
ContextS2-Tests-San			

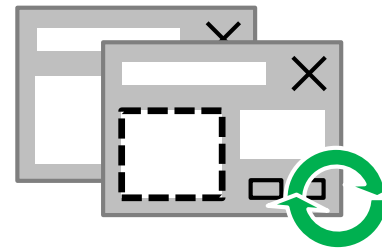
<u>Error: Method not found</u>
fix this

layers here

not here

How to realize?

1. Localize **view-related** source code
2. Express changes **concisely**
3. Update **run-time** components



Updates inconsistent due to mismatch of expected and actual code execution times

Localization difficult due to indirect or missing links from run-time components to code



Modifications verbose due to implicit or explicit duplication or redundancy of unspecific concepts



Research Question

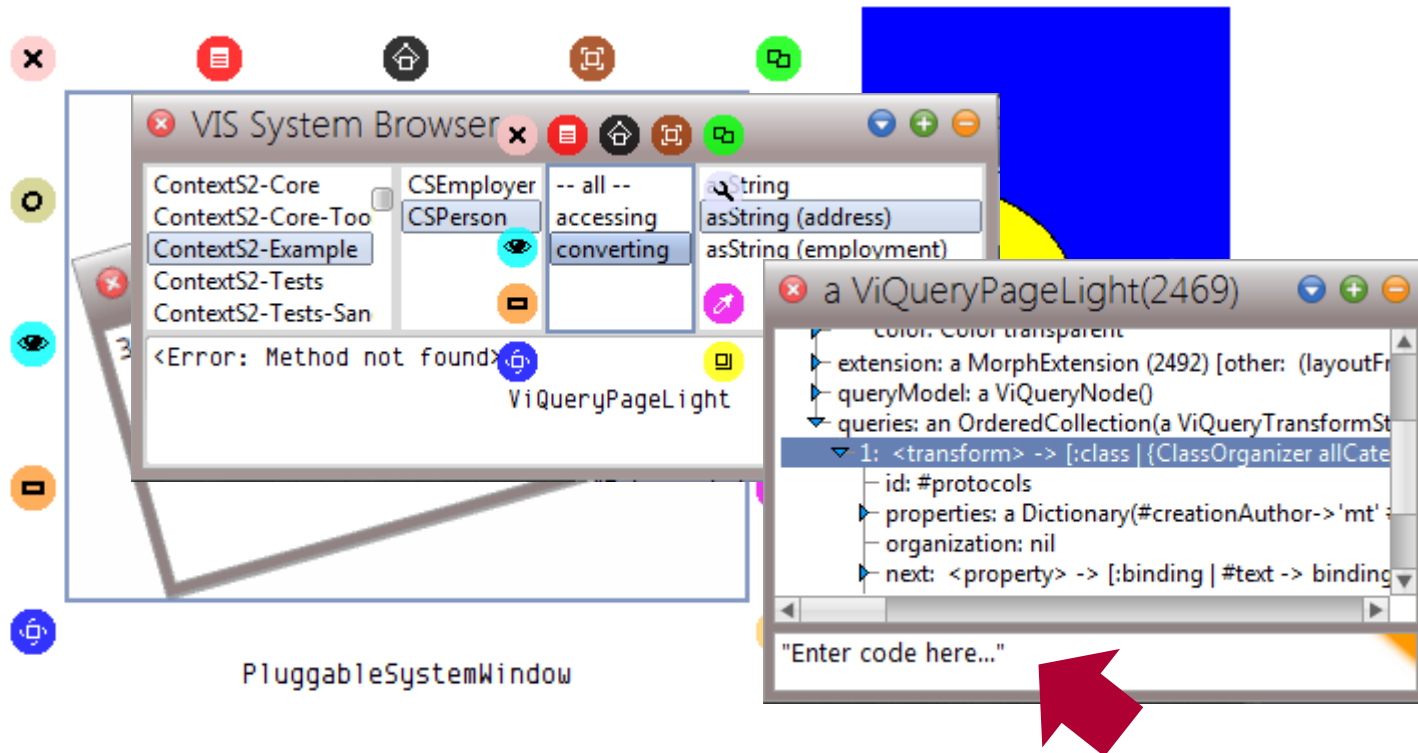
5

How can we support programmers to build programming tools whose specific source code parts can be localized directly and modified concisely while tool instances update consistently?

- Inspect visual elements; localize code
- Adapt code to specific domains, tasks, or users
- Immediately see changes; keep on using the tool
- **Impact?** Support iterative tool building, try out ideas, prototype

Inspecting Visual Elements

6

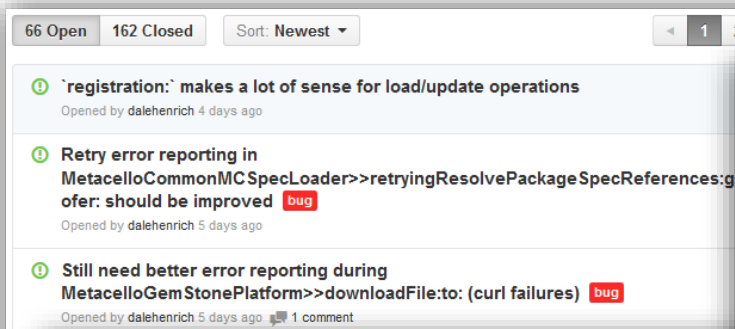


- Works for, e.g., Smalltalk/Morphic, F-Script for Cocoa, ...

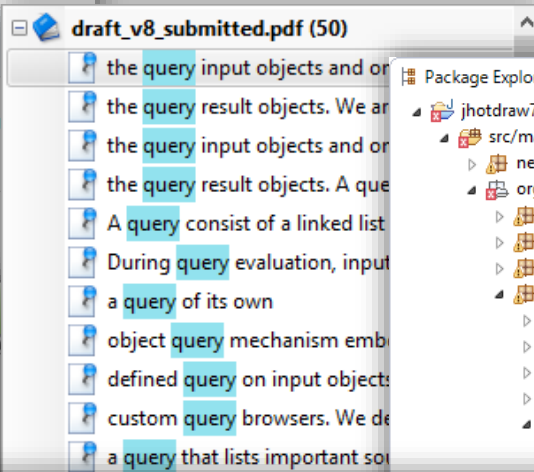
Our Focus: Exploration Tools

7

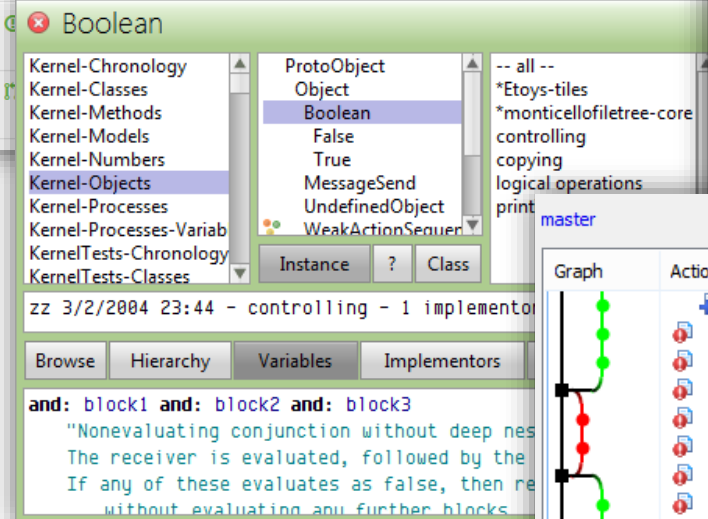
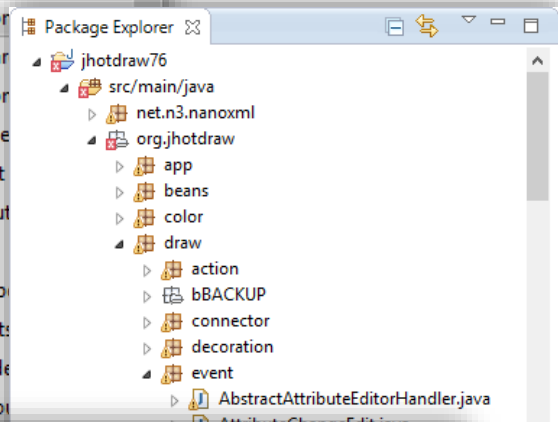
GitHub Issues Browser



Search Result Browser



Package Browser



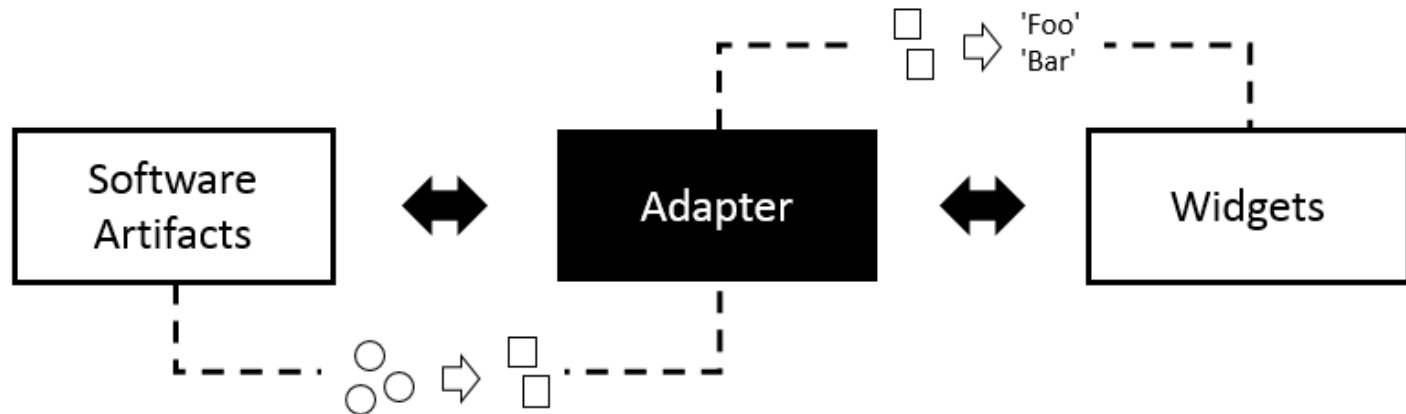
Smalltalk Code Browser



Git Log Browser

Design of Exploration Tools

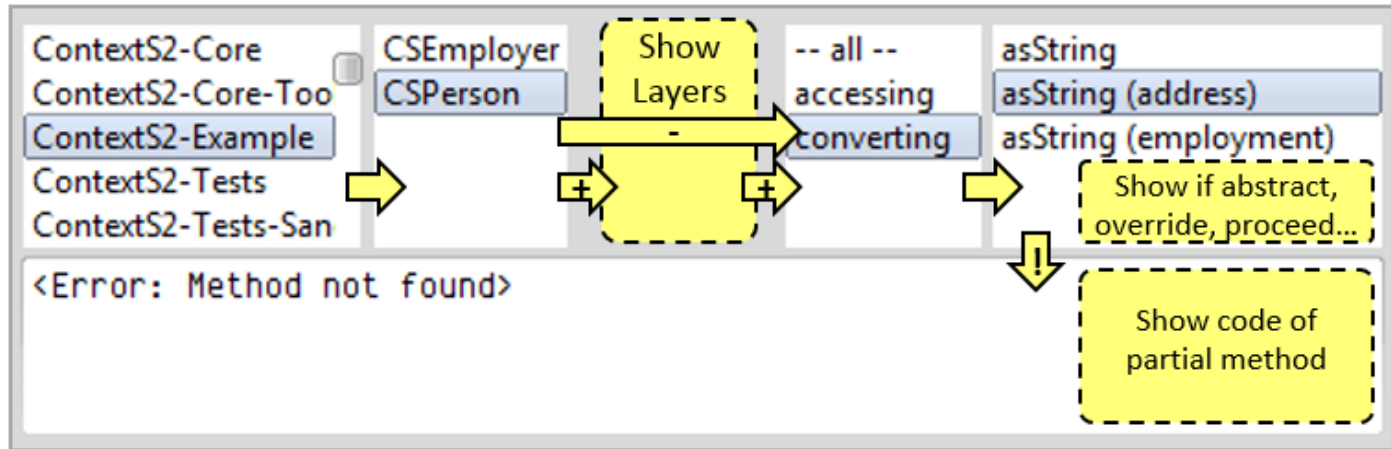
8



- Traditional adapters range from **monolithic** to **composite** designs often with a **declarative** language for tool initialization
- Lessons learned
 - Make specific code declarative (what not how) and reachable
 - Avoid hidden redundancy in declarative code
 - Modularize run-time components to support modular updates

Our Approach: Expose Dataflow in Tools

9



Specific Code in Exploration Tools

10

- Find software artifacts, e.g. all methods of a class:

```
String methodDict values
```

- Read relevant properties of artifacts, e.g. method information:

```
(String >> #findTokens:) selector
```

```
(String >> #findTokens:) getSource
```

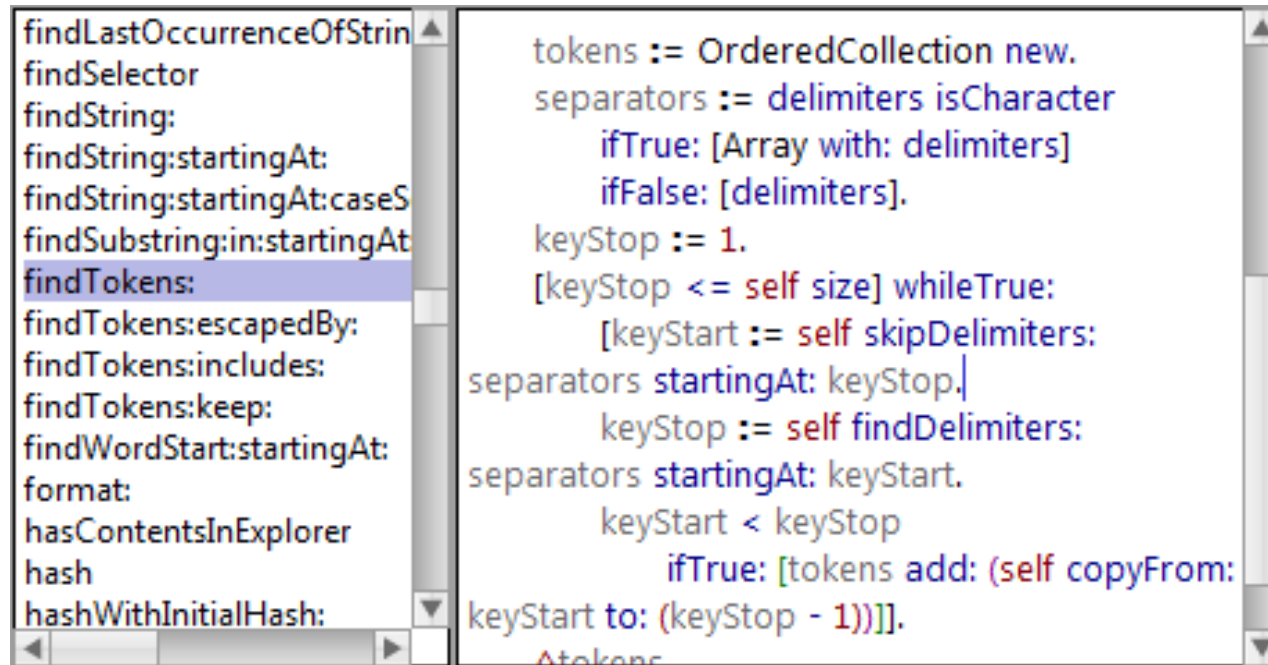
- Show information on screen, e.g. in a text box:

```
ListView position: 0@0 extent: 160@240
```

```
TextBox position: 160@0 extent: 300@240
```

Specific Code in **Exploration Tools**

11



```

findLastOccurrenceOfStrin
findSelector
findString:
findString:startingAt:
findString:startingAt:caseS
findSubstring:in:startingAt
findTokens:
findTokens:escapedBy:
findTokens:includes:
findTokens:keep:
findWordStart:startingAt:
format:
hasContentsInExplorer
hash
hashWithInitialHash:
tokens := OrderedCollection new.
separators := delimiters isCharacter
    ifTrue: [Array with: delimiters]
    ifFalse: [delimiters].
keyStop := 1.
[keyStop <= self size] whileTrue:
    [keyStart := self skipDelimiters:
separators startingAt: keyStop.|
    keyStop := self findDelimiters:
separators startingAt: keyStart.
    keyStart < keyStop
    ifTrue: [tokens add: (self copyFrom:
keyStart to: (keyStop - 1))]].
Atokens
    
```

- Write back changed properties to software artifacts, e.g. code:

```
String compile: 'findTokens: delimiter ...'.
```

"Unspecific" Code to Avoid

12

- Required support structures in frameworks or libraries
 - Subclasses
 - Callback functions
 - ...
- Initialization code
 - Glue callbacks
 - Layout graphical widgets **upfront**
 - ...

Encapsulate Specific Code

13

- **What** information? → Query code

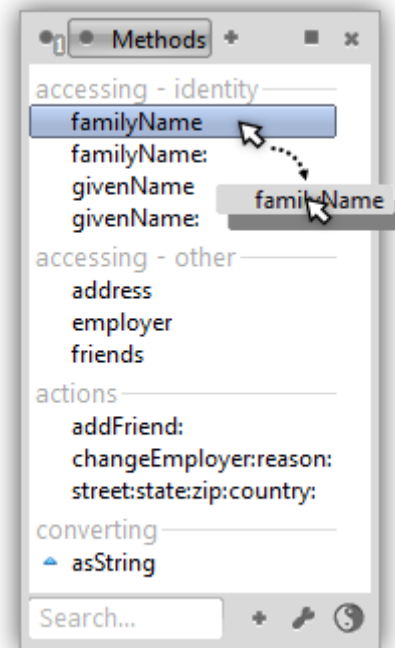
```
[:class | class methodDict values collect: [:method | {
  #text -> method selector.
  #object -> method }]]
```

- **What** graphical output? → View code

```
[:model | ListView new
  model: model;
  textSelector: #text;
  dragSelector: #object]
```

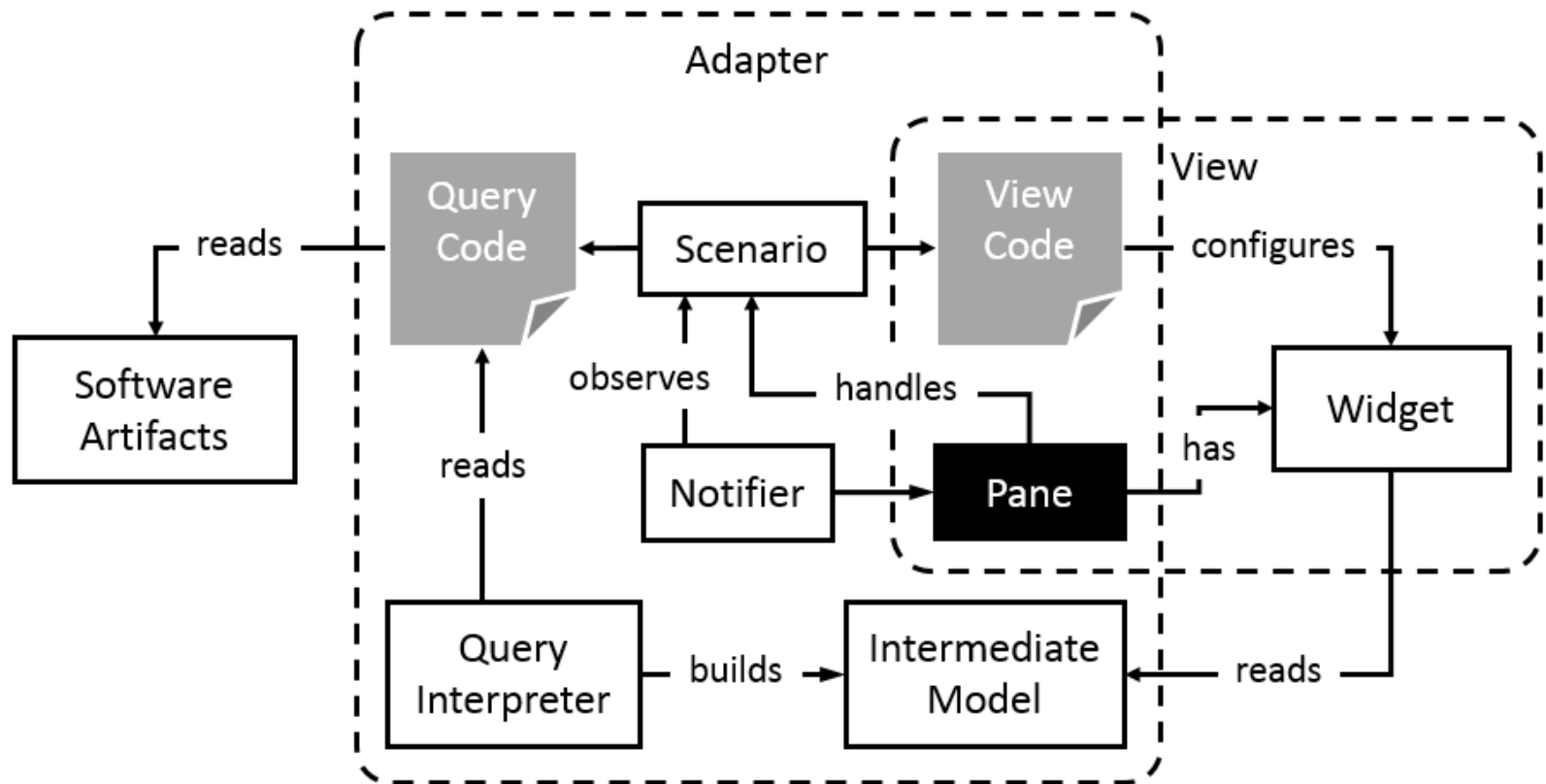
Position and extent to be modified interactively in the programming environment.

- **How** update? → Framework evaluates query code and view code with concrete artifacts automatically.



Our Design for Exploration Tools

14



Modularize Query Code

15

```
[:class | class methodDict values collect: [:method | {
  #text -> method selector.
  #object -> method }]]
```



```
[:class | class methodDict values].
```

```
[:method | #text -> method selector].
```



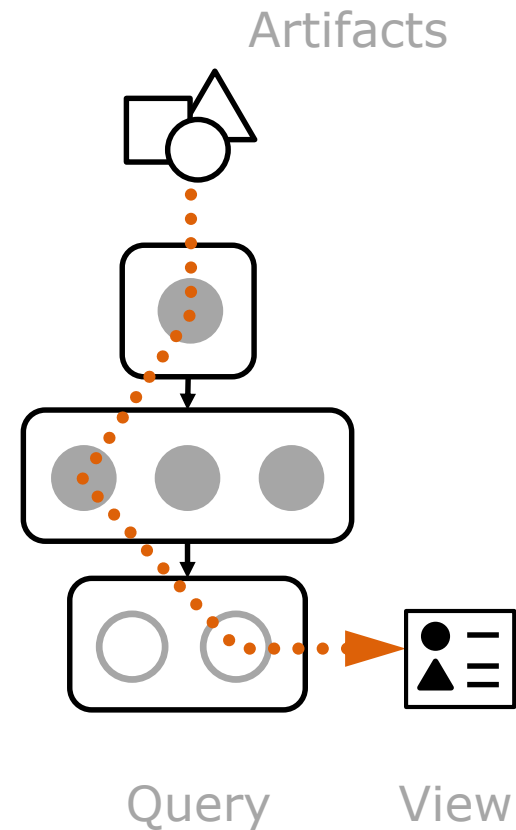
- Provide semantics for handling lists of artifacts
- Decouple artifact transformations from property extractions

Extend Dataflow Metaphor

16

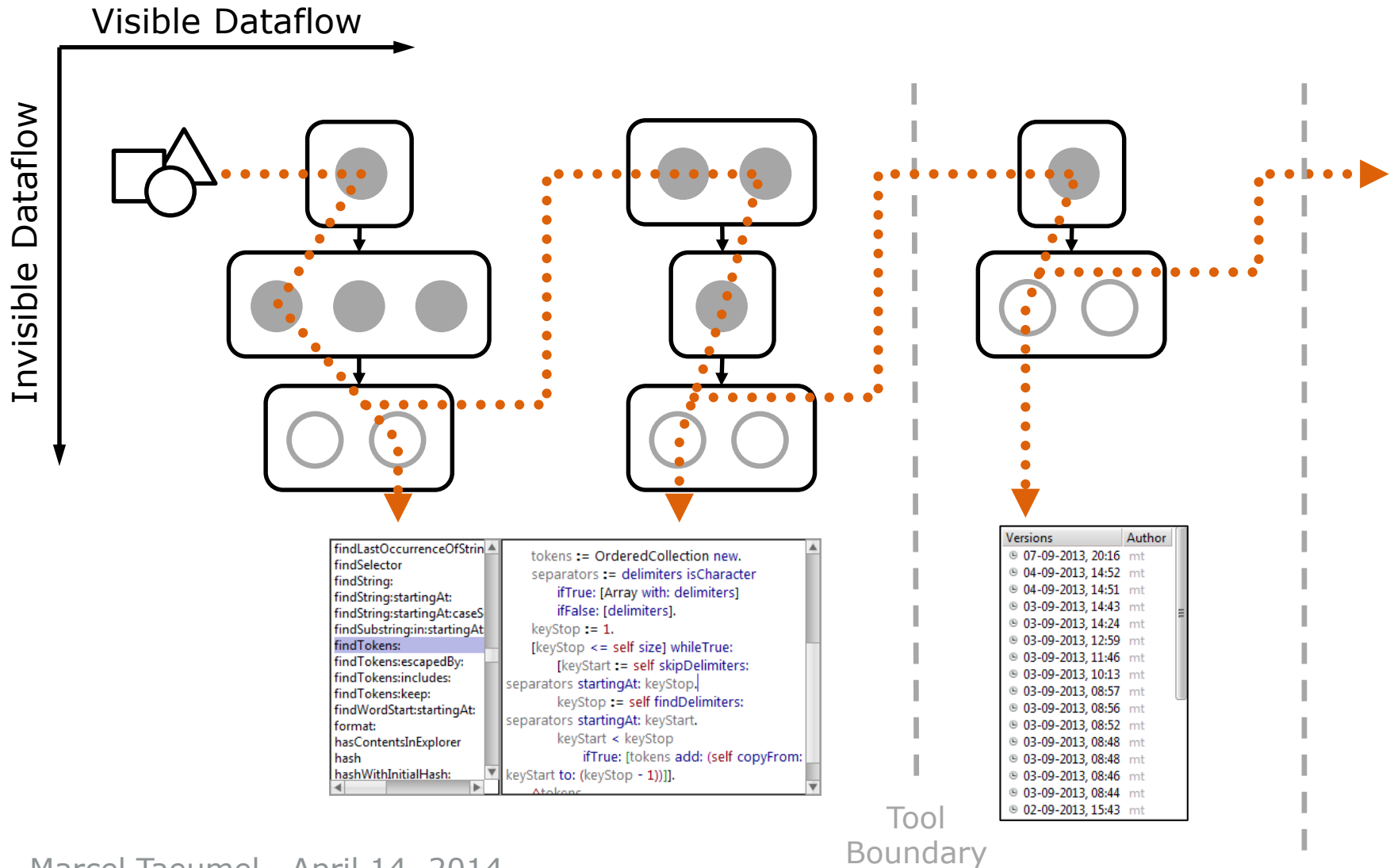
Decouple specific code

- Domain-specific artifact transformation
- Domain-specific property extraction
- Task-specific compositions



Extend Dataflow Metaphor

17

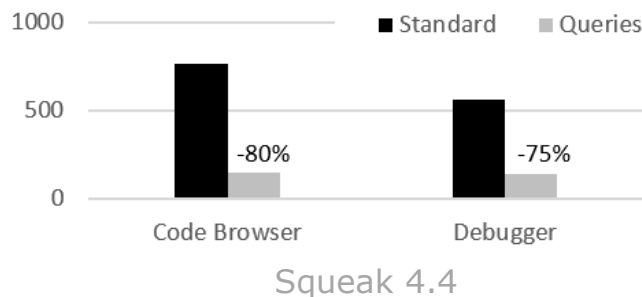
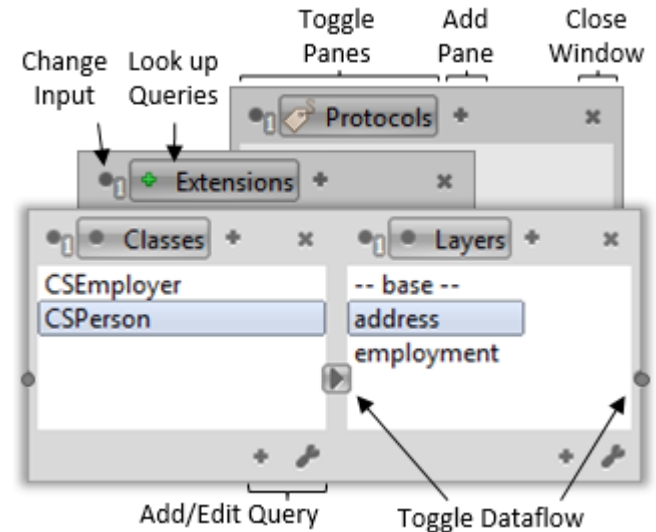


Current and Next Steps

18

Extend **dataflow metaphor**

- Create, update, and delete software artifacts
- Compose, decompose, recommend queries



Evaluate **dataflow metaphor**

- Build programming tools
- Quantify and compare efforts

Conclusion

19

- Programmers are able to iteratively **build** their own tools
- How can we support **localization** and **modification** of code so that tools can be changed in use?
- We employ a **dataflow metaphor** on programming tools

```
ActiveVivide openQueries: {
  [SystemOrganization categories] asQuery.
  [:category | SystemOrganizer default
    classesInCategory: cat] asQuery.
  [:class | ViProtocol protocolsForClass: class] asQuery.
  [:protocol | protocol methods] asQuery.
}
```

