Language and system support for interaction

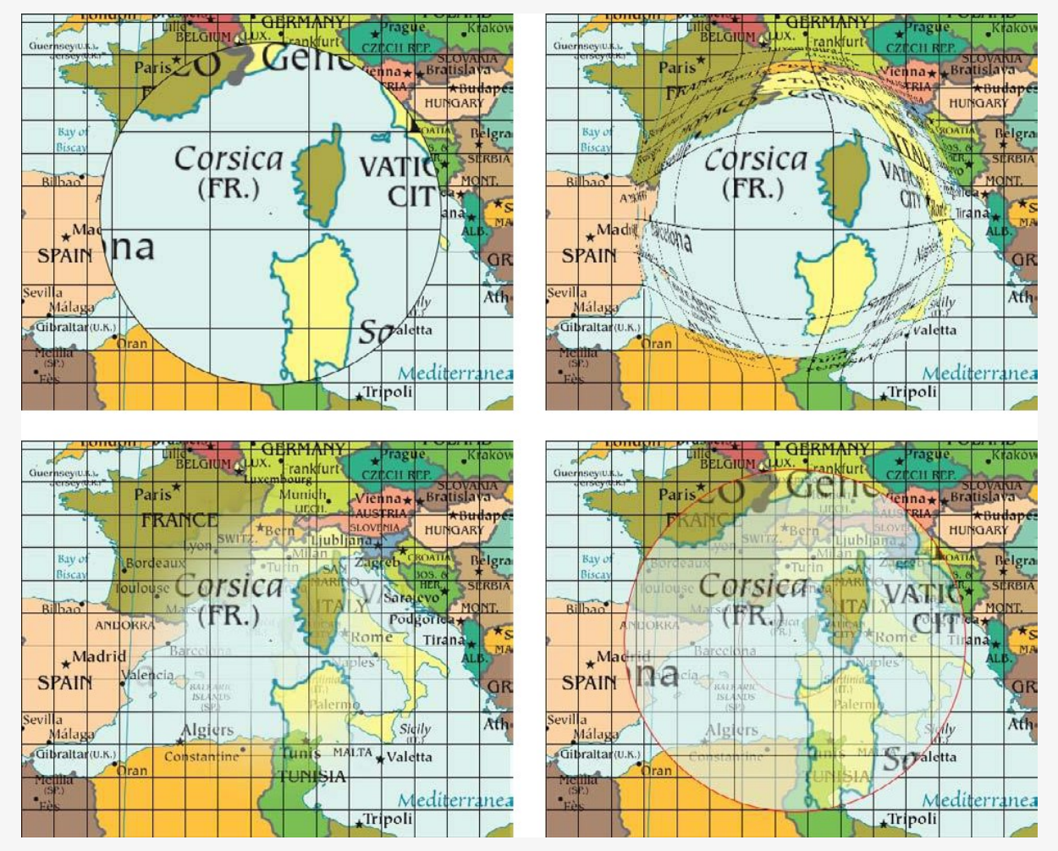
Thibault Raffaillac
Inria, Lille, France
https://traffaillac.github.io

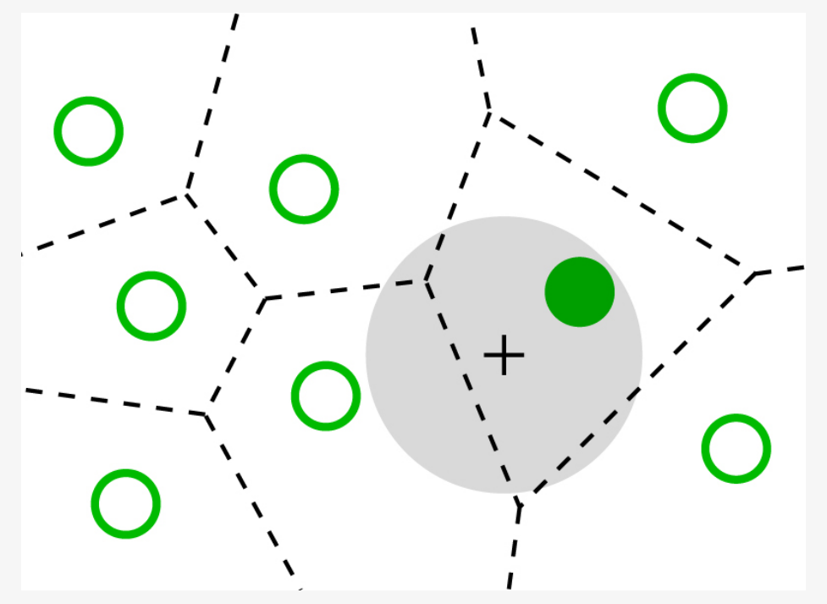
Supporting interaction programming

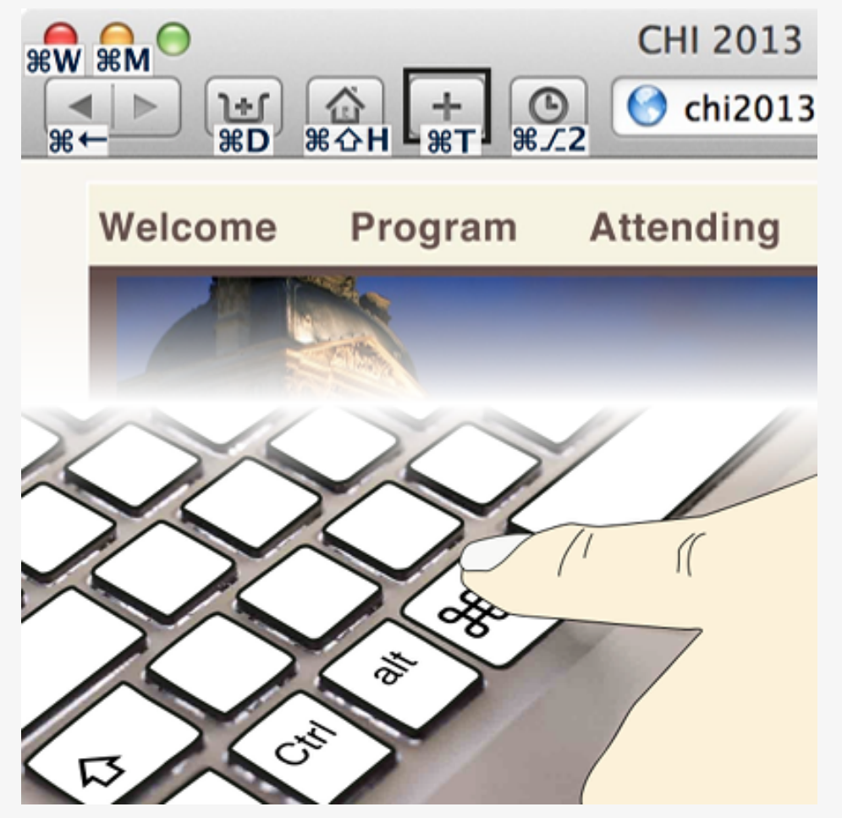
Prototyping **new interaction techniques** is difficult with modern interaction frameworks.

They bring **unexpected scenarios** challenging the frameworks' architectures, like:

- post-filtering the display
- altering the mouse targeting behavior
- highlighting the shortcuts of each icon on screen


Sigma Lenses (Pietriga and Appert)


Bubble Cursor (Grossman and Balakrishnan)



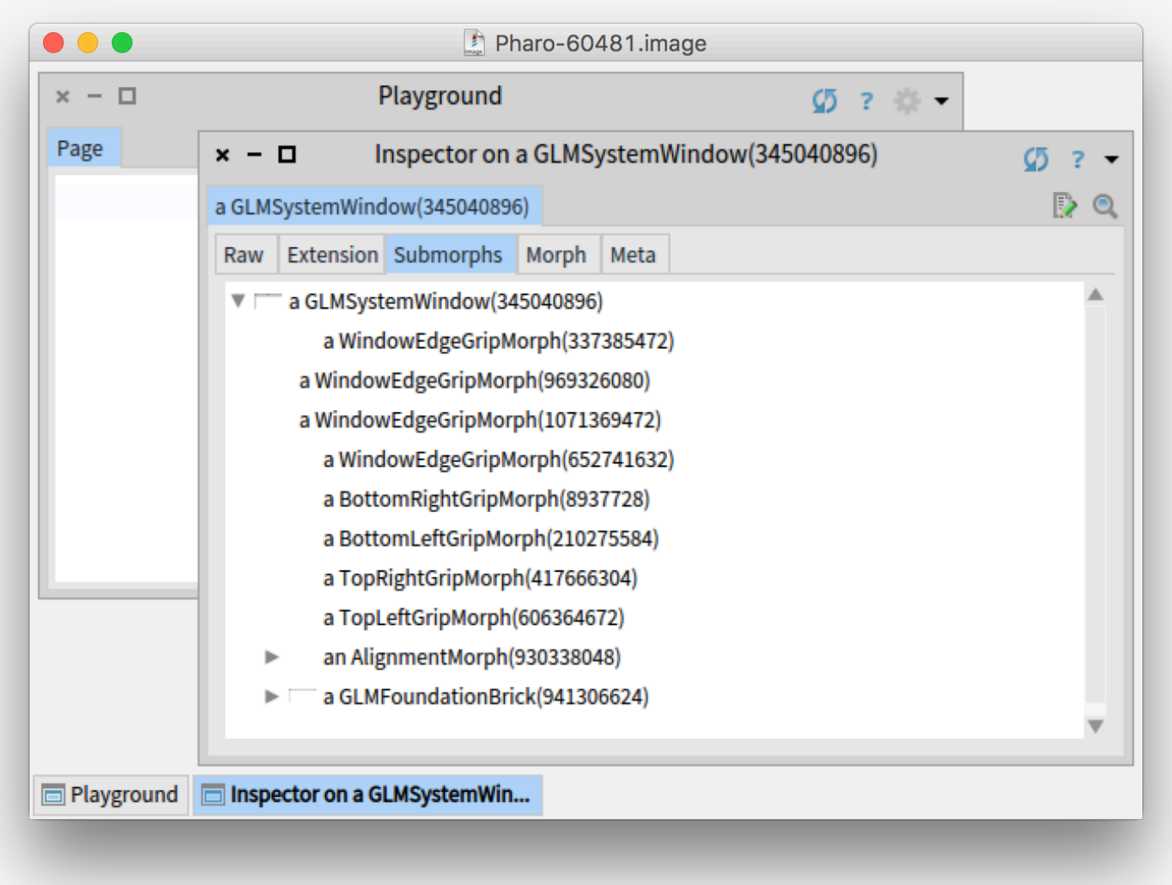
ExposeHK (Malacria et. al)

Researchers and interaction designers must browse hardly-used and ill-documented APIs to access low-level functions. They may mix code from different levels, frameworks, and paradigms. In these conditions, **hacking** is a solution to get a satisfying result in time, but is often not a *perennial* one.

In the context of Pharo Smalltalk

Pharo is a live programming environment, with a pure object-oriented programming language.

It can **introspect** objects in the interface, which we use as a form of *documentation*.


Pharo-60481.image
Playground
Page
Inspector on a GLMSystemWindow(345040896)
a GLMSystemWindow(345040896)
Raw
Extension
Submorphs
Morph
Meta
Playground
Inspector on a GLMSystemWin...

```
▼ a GLMSystemWindow(345040896)
      a WindowEdgeGripMorph(337385472)
    a WindowEdgeGripMorph(969326080)
    a WindowEdgeGripMorph(1071369472)
      a WindowEdgeGripMorph(652741632)
      a BottomRightGripMorph(8937728)
      a BottomLeftGripMorph(210275584)
      a TopRightGripMorph(417666304)
      a TopLeftGripMorph(606364672)
  ▶   an AlignmentMorph(930338048)
  ▶   a GLMFoundationBrick(941306624)
```

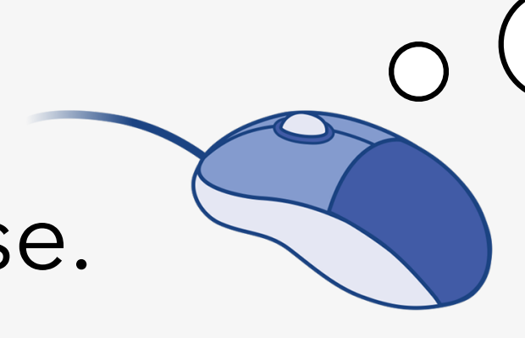
It supports and encourage live prototyping thanks to its **reflective** nature, i.e. one can modify instantiated objects, replace them, and even prototype language extensions.

How can we improve the *flexibility* of interaction frameworks?

Promoting first class objects for interaction

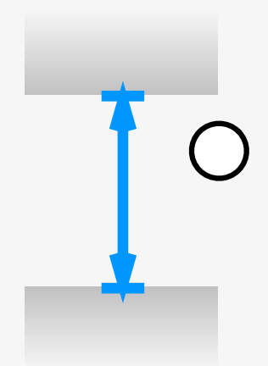
These are objects in the programs (e.g. classes), and also objects in the interfaces (e.g. icons).

They reify *physical* devices, like display, keyboard, mouse.



```
class Mouse {
  float dx, dy;
  bool[] buttons;
  float dpi;
};
```

They may also represent *abstract* elements, like layout constraints, listener links, commands, etc.



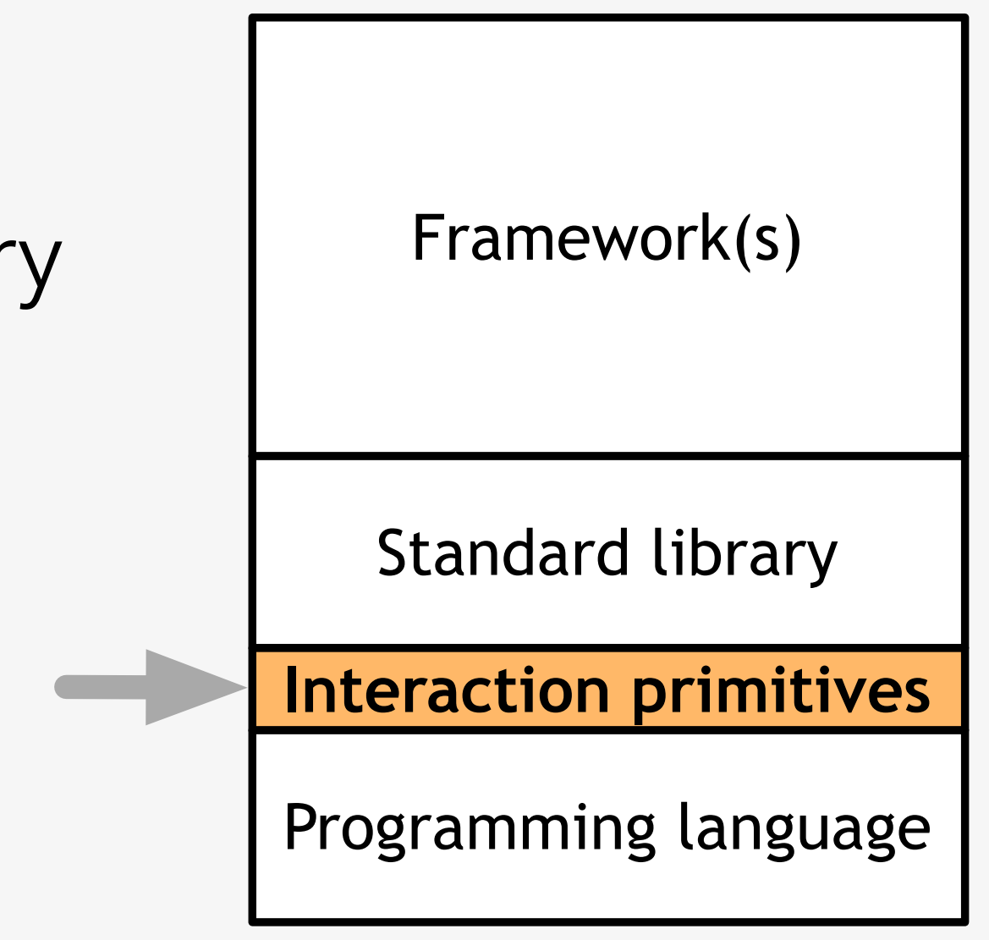
```
class Constraint {
  Widget *from;
  Widget *to;
  float length;
};
```

Such objects are **observable** from their variables, and **modifiable** to alter their effects on interaction. They are also available **across applications**.

Adding primitives to programming languages

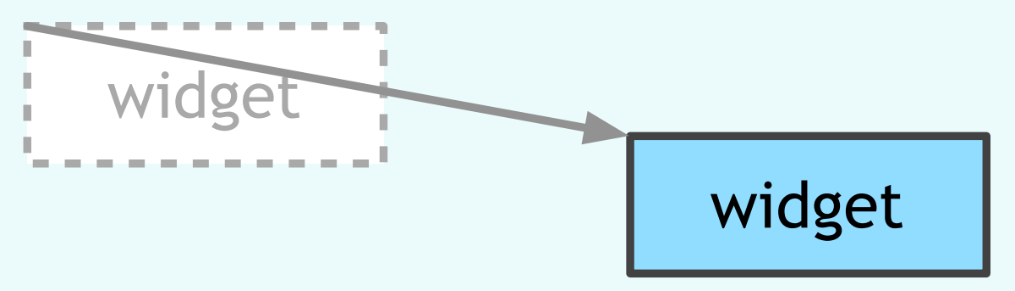
Keywords and functions added to the language or its standard library are available **across frameworks**.

Programming languages enforce **size constraints** on such edits.



They should support existing scenarios, and handle unexpected cases by being **generative**.

```
widget.setPosition(target) during 2s
```



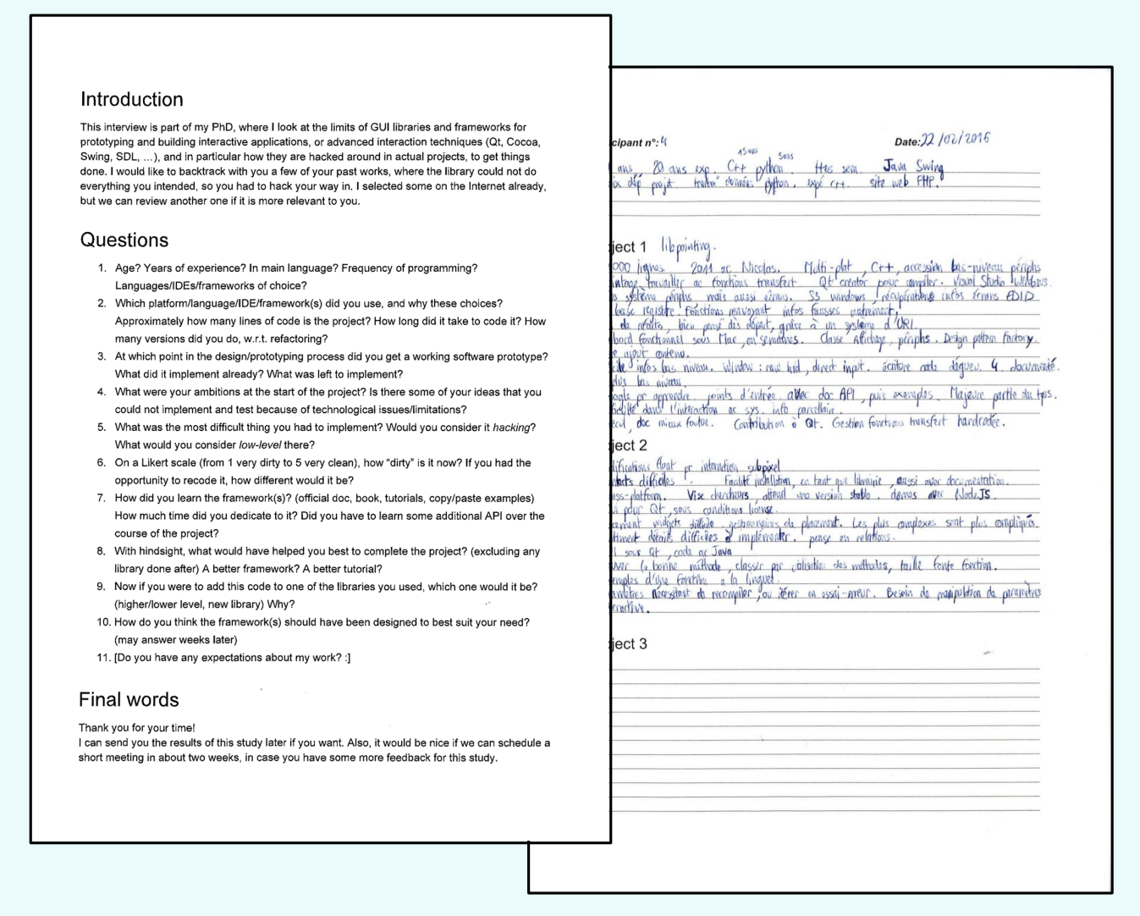
Turning Function Calls Into Animations (EICS'17)

Understanding interaction hackers

Little is known about how people cope with the limits of frameworks when prototyping. Most research informs the design of *documentations* and *APIs*, in contexts of *opportunistic programming* and *learning with examples*.

I conducted 8 interviews of researchers who regularly design and implement new interaction techniques, to gather their:

- Needs
- Problems
- Tools in use
- Hacking strategies



Mjolnir (http://mjolnir.lille.inria.fr/) & *RMoD* (http://rmod.lille.inria.fr) research teams