

TinySelf: Tiny object-oriented language

Code structure and step-by-step guide

Tomas Petricek, Charles University

✉ tomas@tomasp.net

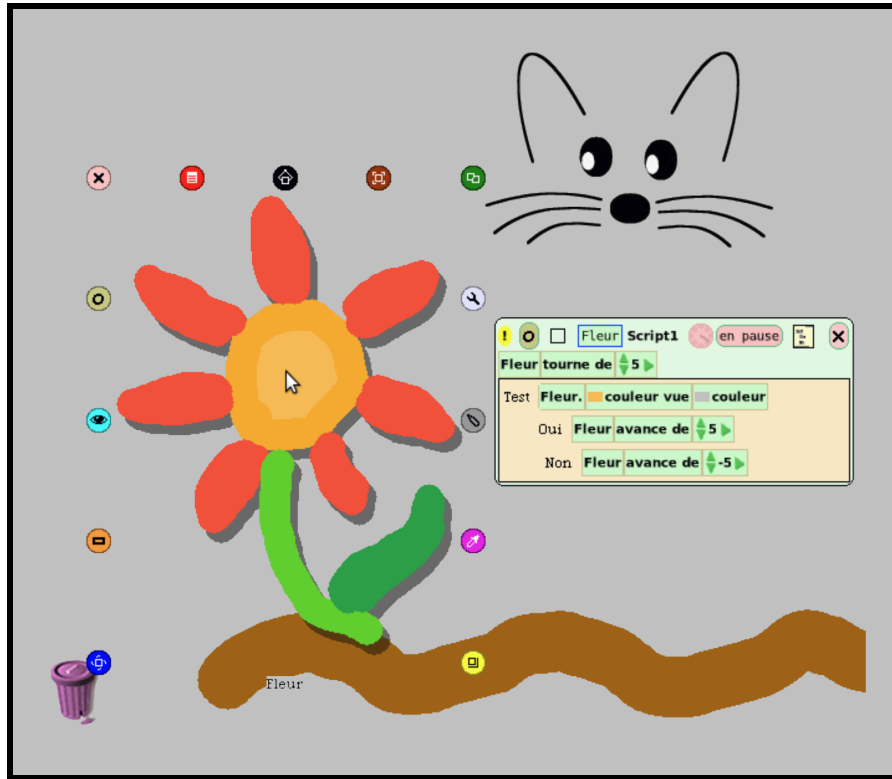
🦋 [@tomasp.net](https://tomasp.net)

🌐 <https://tomasp.net>

🌐 <https://d3s.mff.cuni.cz/teaching/nprg077>



How Self-like systems put things on screen?



Escape hatch is a must

Smalltalk system calls

Self primitive calls

(primitives primitiveList)

TinySelf special objects

Primitive string values

Native F# methods

```

// Special TinySelf objects!
type Special =
  | String of string
  | Native of (Objekt -> Objekt)

// Optionally special object
and Objekt =
  { mutable Code : Objekt option
    mutable Special : Special option
    mutable Slots : Slot list }

// Code to clone an object
let cloneCode =
  { Slots = []; Code = None
    Special = Some(Native(fun arcd ->
      lookupSlotValue "self*" arcd
      |> cloneObject )) }

// Method with special code object
let cloneMethod =
  { Slots = []; Special = None;
    Code = Some cloneCode }

```

Special objects

String values

No other way to
represent strings!

Native methods

F# function taking
"activation record" and
returning the result

Used as method code

Slot lookup logic

Find a set of matching slots

- 1) Search target object
- 2) Search parents and union the results
- 3) Avoid infinite loops!

Input:

obj , the object being searched for matching slots
 sel , the message selector
 V , the set of objects already visited along this path

Output:

M , the set of matching slots

Algorithm:

```
if  $obj \in V$ 
then  $M \leftarrow \emptyset$ 
else  $M \leftarrow \{s \in obj \mid s.name = sel\}$ 
    if  $M = \emptyset$  then  $M \leftarrow parent\_lookup(obj, sel, V)$  end
end
return  $M$ 
```

Where $parent_lookup(obj, sel, V)$ is defined as follows:

```
 $P \leftarrow \{s \in obj \mid s.isParent\}$ 
 $M \leftarrow \bigcup_{s \in P} lookup(s.contents, sel, V \cup \{obj\})$ 
return  $M$ 
```

Message sending logic

Self handbook

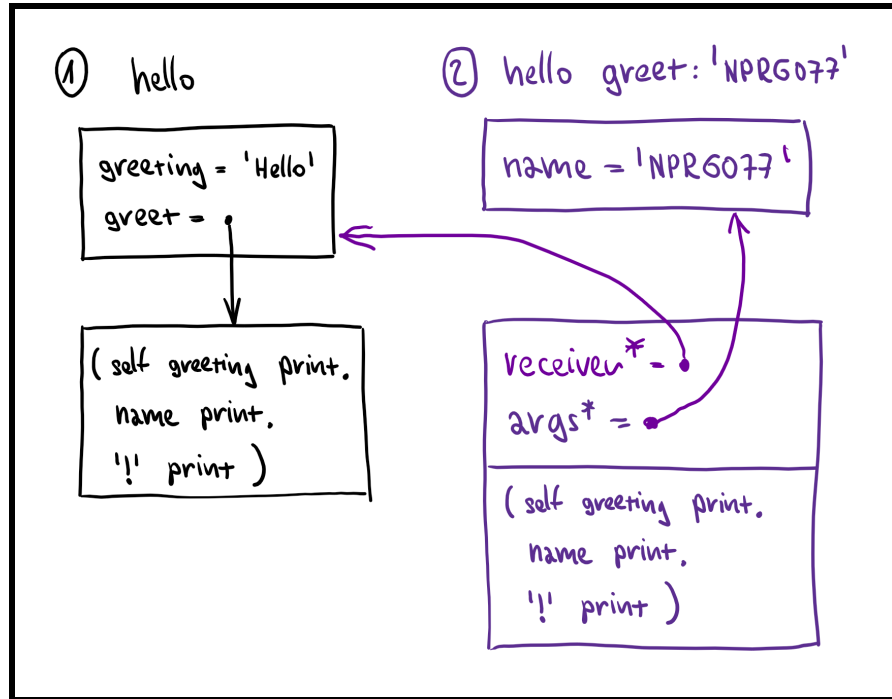
A normal send does a look-up to obtain the target slot;

If the slot contains a data object, then the data object is simply returned.

If the slot contains a method, an activation is created and run.

TinySelf translation

1. Find slot using lookup!
2. Check it is exactly one
3. If there is no code, return it
4. If there is code, run it...
 - Create activation record
 - Run (non-)native code



Activation record

Lookup in activation record to get all our code needs!

Clone of method

It could have data!

Self as parent

Access target's slots!

Arguments as parent

Access arguments!

Sketch

How methods are invoked

Representing TinySelf code

AST is a tree of objects

- Objects store sub-expressions etc.
- Ordinary recursive F# interpreter

```
(self greeting print.  
  name print.  
  '!' print )
```

More object oriented?

- All nodes have **eval** method
- Becomes (a bit) too hard to implement!

Benefits and drawbacks

- Both options differ from actual Self/Smalltalk
- Simpler than actual compiled methods!

Simple expression

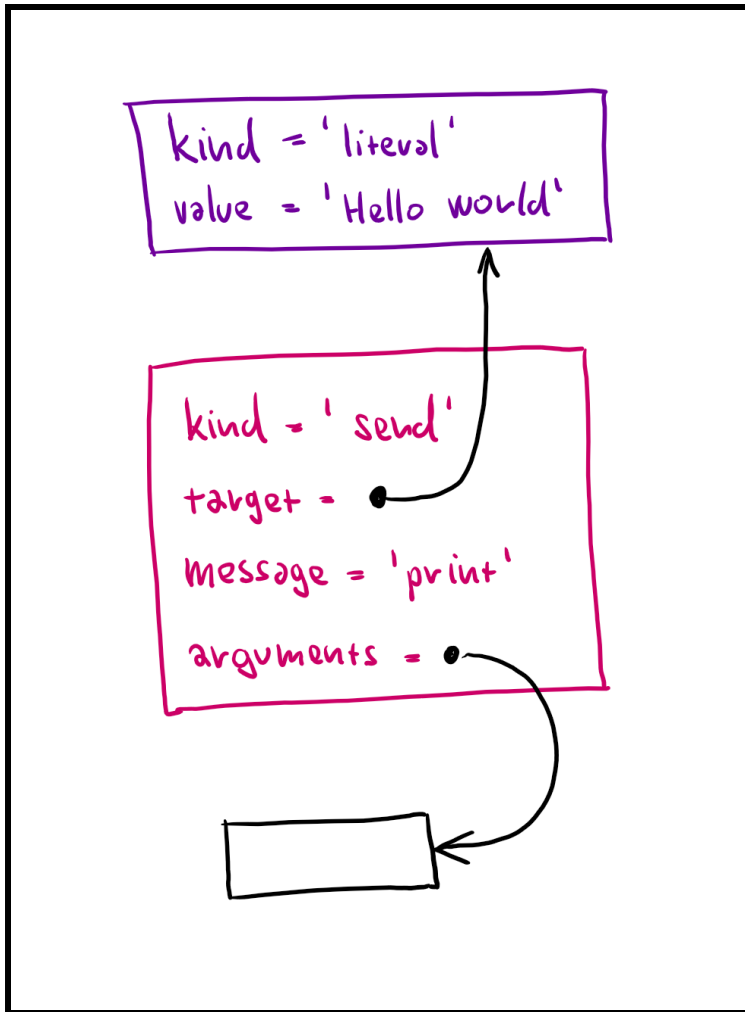
'Hello world' print

Send expression

Receiver, message,
arguments to be used

String expression

String value to be returned



Lab overview

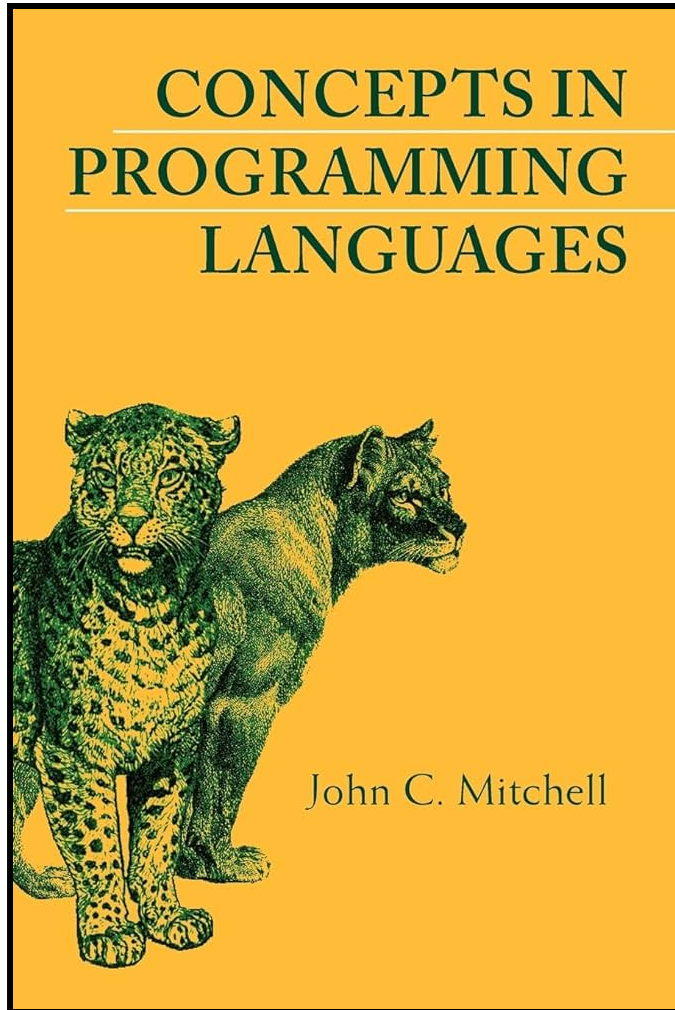
TinySelf system step-by-step

TinySelf - Basic tasks

1. Implementing slot lookup and strings
Traversing the prototype hierarchy to find slots
2. Implementing (basic) message sending
Returning data slots and calling (native) methods
3. Adding method arguments and assignments
Creating assignment slots and revised activation records
4. Object-oriented Booleans and conditionals
Higher-order methods with blocks
5. Representing & interpreting TinySelf expressions
Creating expression objects and an interpreter

TinySelf - Bonus and super tasks

1. Arguments and sequencing of expressions
Adding more types of expressions to TinySelf
2. Revisiting Booleans and conditionals
Representing TinySelf code with conditions
3. Objects as lists and more expressions
Adding more infrastructure before the next step...
4. Creating web-based visualizers
A small step towards TinyMorphic framework



TinySelf and OO

Dynamic lookup

Find method using lookup

Abstraction

No private slots in TinySelf

Subtyping

Object with required slots

Inheritance

By setting a parent slot

Self-sustainable
Complete basic library
Reflection capabilities

Mirror objects
Inspect & modify
Done in Nanospeak



Lessons learned

A tiny prototype-based OO language

-  Basic logic of object-oriented languages
-  Shows how to build self-sustainable system
-  Different implementation - everything is object
-  Hard to implement! Need debuggers, not types