Smalltalk

Yongqi Jia [*], Jiaxu Kang [†]

April 14, 2020

1 Abstract

2 Introduction

3 History

3.1 Why was the language designed?

The purpose of the Smalltalk project is to provide computer support for the creative spirit in everyone. Our work flows from a vision that includes a creative individual and the best computing hardware available. We have chosen to concentrate on two principle areas of research: a language of description (programming language) that serves as an interface between the models in the human mind and those in computing hardware, and a language of interaction (user interface) that matches the human communication system to that of the computer.

3.2 Who designed it?

Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Diana Merry, Scott Wallace, and others

3.3 What is its current status?

While Smalltalk is not "popular" today, it is certainly still widely used.

4 Control Structures

Smalltalk is the first computer language based entirely on the concepts of objects and messages. Therefore, in smalltalk, everything is an object, even 3, true, nil, activation records.[5]

[*]Email: yongqijia@email.arizona.edu
[†]Email: jiaxukang@email.arizona.edu

"Control structures do not have special syntax in Smalltalk. They are instead implemented as messages sent to objects."[3] control structures all handled by message sending. So that's meaning we do not need to built-in control structures. Because of this, we can combine any objects together to create control structures.[4] In terms of that, I'll introduce several objects such as, ifTrue, whileTrue and exception.
ifTrue(same as ifFalse):

```
@condition ifTrue: [— '@temps — '@.statements]
        expr ifTrue: [statements to evaluate if expr]
```

whileTrue:

```
— '@temps —
'@.Statements1.
['index ¡= '@stop]
  whileTrue:
[— '@blockTemps —
 '@.BlockStmts1.
'index := 'index + 1].
 '@.Statements2'
```

Exception:

```
'@block
  on: 'exception
do: [ :'@err — — '@temps — ]
```

This is some codes that I write:

```
IfPrint

x:=0 .

x=1 ifTrue: [ Transcript cr; show: 'this will not printed.' ]

ifFalse: [ Transcript cr; show: 'this will be printed.' ]
```

Everything in Smalltalk is an object. So IfPrint is a method in a class named "P2MyIf". And I give x value first. Then giving some expression before using if function like x=1. ifTrue and ifFalse also are subclasses in known packages. We just call and use it.

5 Data Type

5.1 Smalltalk is Un-Typed

Object-Oriented language like Java, each variable must be declared, and more importantly given a type. Things are different in Smalltalk, every variable must be declared, but it is not given a type. The main reason for that is Smalltalk is consist of objects, and objects can not have a type.

5.2 Declaration

Smalltalk has an unusual syntax for variable declarations: the variables are simply listed enclosed in vertical bars. And also, there is another way to declare variables is that using the key word "instanceVariableNames" and write the variables' name after this key word. For example:

```
int x, y, z;                              // Java
instanceVariableNames: 'a b c'            "Smalltalk"
```

Why bother to declare variables at all, if there is no typing information? The primary answer is that every variable must be declared so that the compiler can set aside space for each variable. Every variable will be allocated 32 bits and can hold either a pointer to an object or a SmallInteger.

5.3 Dynamic Typing

Sometimes, the un-typed language like Smalltalk are said to be "dynamically type", since the compiler does not check the types of variables but each value has a type. Object-Oriented language like Java is said to be "statically type". At compile time, the compiler will checks to make sure every line of the program is type-correct. And at runtime, there is no need to check the type.[1]

5.4 Assignment Statements

The assignment statement is simple in Samlltalk, and Smalltalk always has a simple variable name to the left of the assignment operator. The syntax is:
variable := expression
There are some examples:

```
i := i + 1
x := 8
x := y := z := 7
```

5.5 Type Conversions

In Java, the warp class of primitive data type provides some parse method which could convert the data type. Here are some examples:

```
String a                                  // String
```

int b = String.ParseInt(a) // Int

In Smalltalk, there is only one way to convert data from one representation to another, and this is with message-sending. In Smalltalk, there is a unary message for doing each kind of conversion. The tradition is to give these methods names beginning with "as..." such as "asFloat" or "asString".[1]
instanceVariableNames: 'd i s'
d := i asFloat.
i := d asInteger.
s := i asString.

5.6 Exceptions and Signals

[2] Smalltalk provides a powerful and flexible mechanism to handle exceptions and errors in a graceful way. Errors raise a so called Signal, which can be handled by a handlerblock. Especially with respect to instance creation, deletion and garbage collection, Smalltalk exceptions are very clean and easy to use. Notice: this is an old-style (but still supported) mechanism for instance based exception handling. Signals are more lightweight in that they do not need an Exception class to be defined. I.e. you can create and handle such signals dynamically without the overhead of creating a class for it. Here is the syntax:

1. Signal access:
   - Number arithmeticSignal
   - Number divisionByZeroSignal
   - Number Object errorSignal
2. handling signals/exceptions:
   - a [...some computation ...] on:aSignalOrExceptionClass do:[... handlerBlock ...]
   - aSignalOrExceptionClass handle:[... handlerBlock ...] do:[...some computation ...]
   - aSignalOrExceptionClass catch:[...some computation ...]

Here is a example:

1. To raise an exception:
   - MyException signal.
   - MyException signal: 'With an error message'.
2. To handle an exception:

1 / 0 on: ZeroDivide do: [Transcript showln: 'Oops! Zero divide!'].
1 / 0 on: Error do: [:e — Transcript showln: 'Oops! ' , e className , '!'].

6 Subprograms

7 Summary

8 References

References

[1] Smalltalk: A white paper overview. (Accessed on 04/13/2020).

[2] Smalltalk/x basic classes - exceptions. (Accessed on 04/13/2020).

[3] Smalltalk, Mar 2020.

[4] Adele Goldberg and L Peter Deutsch. Smalltalk. *Encyclopedia of Software Engineering*, 2002.

[5] Fred Rivard. Smalltalk: a reflective language. In *Proceedings of REFLECTION*, volume 96, pages 21–38, 1996.