Language Study, Part I

Martin Kozeny
CSCI 4501: Programming Language Structure
Spring 2011 University of New Orleans

March 29, 2011

1. **a Date of first release of the language.**

   According to [1], Smalltalk-80 system has its roots in the Xerox Palo Alto Research Center starting more than 10 years ago. Because of publishing book in 1982, first release of Smalltalk was in 1972 (Smalltalk-72) (page 3). Some others sources pointed to year 1971.

   b Name of designer or designers.

   As presented in [6], designers of language are Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Scott Wallace.

   c Is the actual language version you are using the original version? an extension of it? a subset of it? neither an extension nor a subset but a version based on the original language? Explain your answer.

   According to [2], Squeak is a modern, open source, fully-featured implementation of the Smalltalk programming language and environment so its version based on on the original language (page ix).

   d If not the original version, state the date of the release of the version you are using, as well as the date of the release of the translator (compiler, interpreter) you are using to write programs.

   On October 1 1996, Dan Ingalls, as he wrote it in [8], annouced to the world existence of the new implementation of Smalltalk called Squeak. I am using interpreter Squeak 3.9 released in October 2006 according to [5].

   e Do the language have an standard? if so, state release date, and state whether the translator you are using implements the standard.

   According to [5], the ANSI Smalltalk standard was approved on May 19, 1998. The official name of the document is ANSI INCITS 319-1998 (R2002). The Squeak Virtual Machine of Squeak broadly follows the specification of chapter 27 of the [3].

   f If the language does not have a standard,

   i list the documents you are using for the release you are using.

   ii how do all the 3 different main language users (language designers, language implementors, and programmers) deal with the understanding of the language features and their semantics without a standard?

   g What programming paradigms does the language support? How well does it support each paradigm? Clearly support your answers with language features.

   Smalltalk supports only object-oriented paradigm according to [3](page 12). As presented in [2], this paradigm is implemented by sending messages to object instead of calling method(page 93). So it not depends on class of the object, but on if the object understand the message. Smalltalk is based on passing messages.

2. **a Describe the component(s) for the formation of an executable program, i.e. what do you need to write a complete program that can be executed?**

   According to [2], for formation of whole program you need to download *Virtual Machine*, *sources* file, that contains the source code for all of the parts of Squeak that don't change very frequently, current system *image* - a snapshot of a running Squeak system, frozen in time(page 3).

   b What language constructs are used for the formation of library units?

   As presented in [2], for the formation of library constructs are used classes. Squeak's root inheritance hierarchy class is *ProtoObject*, which is used to define minimal entities(page 175).

3. **a Evaluate the language in terms of *readability*, *writeability* and *reliability*.**

   (a) *readability* Readability is supported according to [4] by the fact, that Smalltalk is based on few programming concepts(e.g. everything is an object, message passing) and that is an orthogonal language - there are few exceptions compare to other languages(page I). As presented in [2], programmer will need for nearly every application only this data types: *Object*, *Number* and its subclasses, *Character*, *String*, *Symbol* and *Boolean*(page 175).

   (b) *writeability* Writeability is a function of simplicity and orthogonality. Good writeability is also supported according to [4] by the fact, that Smalltalk is based on few programming concepts(e.g. everything is an object, message passing) and that is an orthogonal language - there are few exceptions compare to other languages(page I). Smalltalk provides also good level of abstraction from recursion to polymorphism and encapsulation.

   (c) *reliability* Smalltalk is a fairly reliable language. It provides automated garbage collection, which can help prevent many run-time errors. However the language does not support strong type checking since in Smalltalk is a dynamically-typed language. All of Smalltalk's variables are never declared to have a certain type, nor do they acquire a type once they have a certain kind of data stored in them, however Smalltalk does keep track of the type of data that is being stored in the variables using internal tags. Using reference semantics lead to aliasing and lower reliability - programmer has to be awake.

   b Is it an all-purpose language (sufficiently general)? Support your answer with the existence or lack of language constructs.

   For programming in Smalltalk can programmer use only a few language conctructs. The fundamental construct in Smalltalk is message and as presented before, programmer will need for nearly every application only this data types: *Object*, *Number* and its subclasses, *Character*, *String*, *Symbol* and *Boolean*. That makes Smalltalk very powerful for lot of pragmatics, e.g.: Artificial intelligence reasoning, General purpose applications, Financial time series analysis, Natural language processing, Relational database querying, Application scripting, Internet, Symbolic mathematics, Numerical mathematics, Statistical applications, Text processing, Matrix algorithms.

4. **Give 3 example programs:**
   For the programs you are asked below, avoid writing one page type programs that fail to show the power of the language.

   a One that you can steal, but can also read, understand and explain. Produce an example of its execution (give its input and output.). State clearly the input used in the execution and the output produced.

   Code shown below is algorithm of Insertion sort implemented in Smalltalk by [7]. This algorithm is implemented clasically, array is sorted step by step from array of one element size to array of its whole size. The main idea is to put element into sorted array. Implementation is described in the source code below in detail. This method was put to the list of methods of class 'Array' in 'Collections-Arrayed' category.

```
insertionSort: sortFun
        "Sort self according to 'sortFun'"
        | el j |                                    "local variables − actual element and actual index"
        self                                        "reference to array itself"
                doWithIndex: [:elem :index |  "indexing array"
                        el := self at: index.   "variable el is initialized to first element of array"
                        j := index − 1.         "variable j is initialized to the end of sorted array, at the beginning to 0"
                        [j >= 1
                                and: [(sortFun
                                                value: (self at: j)
                                                value: el) not]]  "while j index is not lower than 1"
                                whileTrue: [self                   "and actual element of sorted"
                                                at: j + 1          "array is not lower than element to be put into array,"
                                                put: (self at: j). "actual element is shifted right and j index is decreased"
                                        j := j − 1].
                        self at: j + 1 put: el]                    "after putting element to appropriate place,"
                                                                    "to el is put next element behind actually sorted array"
```

1 Tests

Here are some tests of the algorithm and their outputs.

1.1 Test 1

```
a := Array new: 5.
        a at: 1 put: 2.
        a at: 2 put: 7.
        a at: 3 put: 0.
        a at: 4 put: 5.
        a at: 5 put: 3.
a insertionSort: [ :x :y | x < y ].
a do: [ :item | Transcript show: item; cr ]
```

Output:
0
2
3
5
7

1.2 Test 2

```
a := Array new: 7.
        a at: 1 put: 12.
        a at: 2 put: 7.
        a at: 3 put: 10.
        a at: 4 put: 32.
        a at: 5 put: 77.
        a at: 6 put: 43.
        a at: 7 put: 13.
a insertionSort: [ :x :y | x < y ].
a do: [ :item | Transcript show: item; cr ]
```

Output:
7
10
12
13
32
43
77

1.3 Test 3

```
a := Array new: 10.
        a at: 1 put: 2.
        a at: 2 put: 27.
        a at: 3 put: 140.
        a at: 4 put: 312.
        a at: 5 put: 72.
        a at: 6 put: 41.
        a at: 7 put: 32.
        a at: 8 put: 67.
        a at: 9 put: 71.
        a at: 10 put: 89.
a insertionSort: [ :x :y | x < y ].
a do: [ :item | Transcript show: item; cr ]
```

Output:
2
27
32
41
67
71
72
89
140
312

b Write one that is 'representative' of the language application area. Make it a meaningful program.

I wrote as a 'representative' application simple game called Quinto, which shows how easy is to implement in Squeak game with GUI. The game board consists of rectangular array of light yellow cells. When you click on one of the cells with the mouse, the four surrounding cells on the north, south, west and east turn blue. Click again, and they toggle back to light yellow. The object of the game is to turn blue as many cells as possible. Source code of the game implementation is shown below.

```
SimpleSwitchMorph subclass: #SBECell
        instanceVariableNames: 'mouseAction'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'SBE--Quinto'!

!SBECell methodsFor: 'accessing protocol' stamp: 'MK 3/25/2011 16:17'!
mouseAction: aBlock
^ mouseAction := aBlock! !

!SBECell methodsFor: 'initialization' stamp: 'MK 3/26/2011 23:28'!
```

```
initialize
  super initialize.
  self label: ''.
  self borderWidth: 2.
  bounds := 0@0 corner: 16@16.
  offColor := Color paleYellow.
  onColor := Color paleBlue darker.
  self useSquareCorners.
  self turnOff
! !

!SBECell methodsFor: 'event handling' stamp: 'MK 3/25/2011 16:24'!
mouseUp: anEvent
mouseAction value
! !

BorderedMorph subclass: #SBEGame
        instanceVariableNames: 'cells'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'SBE--Quinto'!

!SBEGame methodsFor: 'as yet unclassified' stamp: 'MK 3/26/2011 23:39'!
cellsPerSide
"The number of cells along each side of the game"
^ 10! !

!SBEGame methodsFor: 'as yet unclassified' stamp: 'MK 3/29/2011 10:54'!
newCellAt: i at: j
"Create a cell for position (i,j) and add it to my on--screen
representation at the appropriate screen position. Answer the new cell"
| c origin |
c := SBECell new.
origin := self innerBounds origin.
self addMorph: c.
c position: ((i-1) * c width) @ ((j-1) * c height) + origin.
c mouseAction: [self toggleNeighboursOfCellAt: i at: j].
^ c
! !

!SBEGame methodsFor: 'game logic' stamp: 'MK 3/29/2011 10:57'!
toggleNeighboursOfCellAt: i at: j
(i > 1) ifTrue: [ (cells at: i - 1 at: j ) toggleState].
(i < self cellsPerSide) ifTrue: [ (cells at: i + 1 at: j) toggleState].
(j > 1) ifTrue: [ (cells at: i at: j - 1) toggleState].
(j < self cellsPerSide) ifTrue: [ (cells at: i at: j + 1) toggleState].
! !

!SBEGame methodsFor: 'initialization' stamp: 'MK 3/29/2011 10:54'!
initialize
| sampleCell width height n |
super initialize.
n := self cellsPerSide.
sampleCell := SBECell new.
width := sampleCell width.
height := sampleCell height.
self bounds: (5@5 extent: ((width*n) @(height*n)) + (2 * self borderWidth)).
cells := Matrix new: n tabulate: [ :i :j | self newCellAt: i at: j ].
! !
"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!
SBEGame class
        instanceVariableNames: 'height width n c'!
```

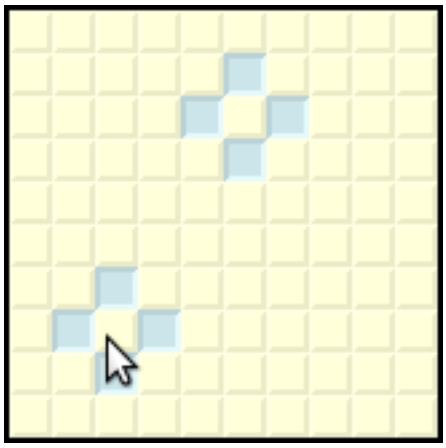

Figure 1: Quinto game at the beginning

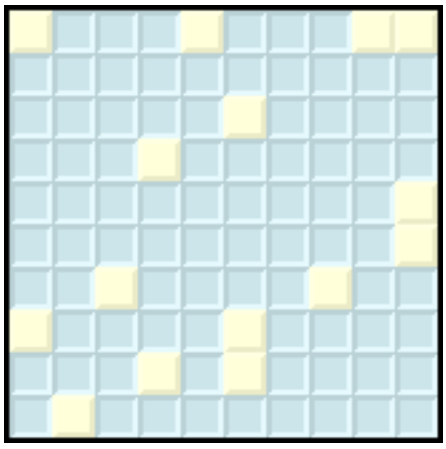

Figure 2: Quinto game execution

References

[1] Glenn Krasner; *Smalltalk-80, Bits of History, Words of Advice*, 1982.

[2] Andrew P. Black, Stéphane Ducasse, Oscar Nierstrasz, Damien Pollet; *Squeak by example*, 2009.

[3] Adele Goldberg, David Robson; *Blue Book*, 1983.

[4] Canol Gökel; *Computer Programming using GNU Smalltalk*, 2009.

[5] Squeak website, `http://www.squeak.org`.

[6] Wikipedia, The Free Encyclopedia, `http://en.wikipedia.org/wiki/Smalltalk`.

[7] Literate programs, Insertion sort;

`http://en.literateprograms.org/`
`Insertion_sort_%28Smalltalk%29#chunk%20use:insertionSort.st`

[8] Dan Ingalls;

`http://groups.google.com/`
`group/comp.lang.smalltalk/browse_thread/thread/`
`798b7a065f08bd7f/70e0d90098d816ef#70e0d90098d816ef`