Chun Chung Cheung
CIS24 SA
4/7/2003
Report #2
Prof Kopec

# Smalltalk Programming Language

The Smalltalk is a pure object-oriented programming language, because everything from items as simple as the integer constants to large complex software systems are objects. Smalltalk is the most mature object-oriented language on the market. Most other object-oriented language implementations, ranging from Object COBOL to Java, have drawn inspiration from Smalltalk.

The idea of Smalltalk development was originated in the Ph.D. dissertation work of Alan Kay in the late 1960's at the University of Utah (Kay, 1969). Kay is one of the fathers of the idea of the object-oriented programming. He believed that non-programmers would use desktop computer widely, therefore, a powerful human interfacing capabilities of programming language would be needed. Kay determined computers would have to be highly interactive and sophisticated graphics interface for users. Then he combined ideas of Flex, ALGOL 60 and SIMULA 67 programming languages to develop Smalltalk.

In 1972, Kay envisioned a system, Dynabook, in which was a general information processor and based on the Flex language to develop. Flex was using a concept of screen windows, users would interact with such a display both through a keyboard and by touching the screen with fingers to get information and graphics. Kay found his way to the Xerox Palo Alto Research Center (Xerox PARC) and presented his

idea on Dynabook, and then he developed the Xerox Alto hardware and the Smalltalk-72 software. Along with experiments and further developments, Smalltalk led to a sequence of languages that ended with Smalltalk-80. In 1987, a full graphical version of Smalltalk/V that supported MS-DOS was released. Then different versions of Smalltalk were released and they supported different platforms, such as, Unix, Windows and OS/2. ANSI standard for Smalltalk was finalized in 1998.

The program units of Smalltalk are **objects.** An object may be composed of other objects, which determine how it behaves. For example, a car is composed of body, engine, suspension, etc., and these in turn are also composed of simpler objects. Basically, objects are constructed by local data and collection of operations called **methods**. All objects have local memory inherent processing ability, the capability to communicate with other objects, and the inherit characteristics from ancestors. In addition, methods are like function definitions and capable of returning values after completed with parameters. There are several examples of methods to illustrate the use of temporary variable and a looping using a method **whileTrue** and block.

**Figure 1.**

**Smalltalk version**

```
MyAdd = aNumber
        | result |
result := aNumber+10.
^result.
```

**C version**

```
Myadd(int x)
{       int result;
        result = x+10;
        return result;
}
```

**Figure 2.**

```
count <– 1.
sum <– 0.
[count < = 200]          " The block with the loop condition"
        whileTrue: [sum <– sum + count.
        count <– count + 1]      " The loop body "
```

**Figure 3.**

**Smalltalk version**

```
(count<100)
        ifTrue:[count:=count+1]
        ifFalse:[Transcrpt show:
                'Overflow'.
                Transcrpt cr.
                Count:=0.]
```

**C version**

```
if(count<100)
                Count++;
else{
        Printf("Overflow\n");
                Count=0;
}
```

In Figure 1, this method adds 10 to the value of parameter and place in the result, which is an instance variable. The value of return is the returned object. The other code is using a C to write that is equivalent to the code of Smalltalk. In Figure 2, the code shows how the class block provides a condition control and pass Boolean objects, true or false to the second block through method whileTrue. There are used a full stop to separate each statement. Figure 3, this is another example to show the difference between C and Smalltalk.

The syntax of Smalltalk is simple and regular. It contains many object-oriented programming features, such as, reuse, encapsulation, inheritance, and polymorphism. In order to increase the productivity in software development, the feature of reuse code is developed in Smalltalk. It also contains encapsulation. Objects reveal their "outside" through tangible behavior, but keep their "inside" hidden. Users do not need to know how

objects work but they need to know how it behaves. Moreover, Smalltalk system includes a large hierarchy of classes. The program consists of creating subclasses from existing classes. Subclasses inherit all of the instance variables, instance methods and class methods of superclass. Polymorphism is the ability of objects to send the same message to instance of different classes, which do not have a common superclass.

Finally, the concept of class hierarchy, reuse, objects, and message passing are powerful for Smalltalk object-oriented program development. Smalltalk also provides dynamic binding that allows type errors undetected until run time. This is a great deal of error repair later in the development than would occur in static typed language. Since the Smalltalk provides virtual machine with byte code interpreter, it can be ported to any platforms. The compiling time of Smalltalk programs are slower when compare with imperative language programs. However, the interface of Smalltalk has had an important impact on computing. The integrated use of windows, mouse pointing devices and pop-up and pull-down menus dominate contemporary software systems. It also inspires computer experts to enhance version of object-oriented programming language, such as, C++, Ada95, and Java.

References:

Budd, Timothy. A Little Smalltalk.
US: Addison-Wesley Publishing Company, Inc.1987.
Robert W. Sebesta. Concepts of Programming Languages. 3rd ed.
CA: Addison-Wesley Publishing Company, Inc.1996.
John Hunt. Smalltalk and Object Orientation: An Introduction.
UK: Springer Verlag. 1997.