**Scala application using AKKA by *Marta Doberschuetz***

1. Introduction

The hardware has been developing very quickly in the past few years, in fact so quickly that the software has not been able to keep up with it. The most popular programming languages like Java or C# cannot use multicore processor to the fullest of its abilities. That’s why it has become very important to create languages that would be able to cope with the modern hardware.

One of those languages is Scala, created by Martin Odersky, a German computer scientist.

1. Programming paradigms

A paradigm, in science, is defined as a set of concepts or thought patterns in a given discipline. (ref. Odersky – 1st course). In computer science a programming paradigm is a way of creating the structure and components of computer applications. Computer scientists distinguish between several programming paradigms, the main ones are:

* imperative
* functional
* object-oriented
* logic
* symbolic.

Some, like Martin Odersky, the creator of Scala, name only the first 3 as main programming paradigms and claim for the object-oriented to be orthogonal to the 3 paradigms and one that combines the features of the 3 paradigms.

Imperative programming uses statements for computation. Statements change the state of the program. In imperative programming, the mutable variables are modified, assignments are used, and so are control structures such as if-then-else, loops, break, continue, return. Programs written using imperative programming paradigm specify a list of tasks that the computer is to perform. Imperative programming indicates how the program should achieve the end result (in what sequence the tasks should be executed) and not what the program should accomplish, like in declarative programming, which is the opposite to imperative programming. Functional and logic programming are examples of declarative programming.

Functional programming uses functions to perform calculations. It does not use mutable variables, assignments, loops and other imperative control structures. In functional programming functions can act like values and those values can be produced, consumed and composed. Additionally functions can be defined in other functions, they can be passed as parameters and returned as results.

Logic programming uses mathematical logic to create programs. Programs written using programming paradigm consist of sequences of logical statements that present facts and rules of the domain problem and an inference algorithm.

Object-oriented programming uses objects to describe the current world state. Objects are described by data fields and the behaviour of the objects is described by methods. Objects then, as instances of classes, interact with one another to create applications and programs.

1. Functional languages

<http://en.wikipedia.org/wiki/Functional_programming>

1. Object-oriented languages
2. Scala
   1. Scalability

The name “Scala” stands for “scalability”. The term scalability means for the network, process or system to be able to manage when the demands of the users grow and effectively grow with those demands. (Book).

Scala runs on a Java platform and uses Java libraries. Scala combines functional and object-oriented concepts. One can only imagine what benefits functional simple function construct and object-oriented able –to-manage-large-systems construct can bring. The combination of the two programming styles also makes the code more concise. (Book). Let’s look at the two code snippets to visualise it:

//enter code here

* 1. Functional and object-oriented language

In Scala, the value of a function is an object! (Book, page 45). But first things first.

In object-oriented programming, objects structure programs. They are “containers” for both data and operations performed on that data. Object are also values themselves and can be stored in other object or passed as parameters in operations. (book, page 45). Although object-oriented languages are based on objects, they also have such thing as primitive data types (like in Java), that are not objects and are used to store values, or some methods do not belong to any objects. All of that seems to limit scalability. Whereas in Scala there is no such limitation as everything is an object: every value, every operation. (Book, page 46). For example, if you perform a calculation like: 1+2 in Scala, you actually invoke a method called: + from class Int. (Book, page 46).

In functional programming, functions are first-class values, which means they have the same status as primitive data types or String objects. Functions can be passed as arguments to other functions, functions can be returned as a result or stored in variables. A function can be defined inside another function same way an integer variable can be defined inside a function. (Book, page 47). “Functions that are first-class values provide a convenient means for abstracting over operations and creating new control structures. This generalization of functions provides great expressiveness, which often leads to very legible and concise programs. It also plays an important role for scalability.” (Book, page 47).

“Immutable data structures are one of cornerstones of functional programming.” (Book, page 48). In Java strings are objects of a class String, they are immutable, which means that when you create an object String and then you want to change that object, what you in fact do, is you create another object. If we just look at strings in Java we can say that Java is a functional language. In this case Scala is like Java – it treats strings in a mathematical sense and not as an array of characters – Scala does not allow mutability. Scala also introduces immutable lists, tuples, maps and sets.

Another characteristic of a functional language is the fact that its methods should have no side effects. If a function or expression has a side effect it means that not only does it return a value, it also modifies state or it interacts with the outside world. Functional languages encourage methods with no side effect, Scala gives the programmer a choice: he/she can either write in an imperative style (with mutable data and side effects) or he/she can avoid it as Scala makes it easy to do so.

Methods with no side effects are called: referentially transparent, which means that a method call can be replaced by its result and the program’s semantics will not be affected. (Book, page 48). In fact, if an expression in Scala has no side effects it can be evaluated using a method called a “substitution model”, which idea is to reduce an expression to a value.

* Parallelism
* concurrency

1. AKKA
2. Application
3. References:

* <http://www.drdobbs.com/architecture-and-design/interview-with-scalas-martin-odersky/231001802>
* Wikipedia