The Tea™ Scripting Language: An Overview

Jorge Nunes *João Paulo Luis*
Luis Miguel Campos

PDM&FC

jorge.nunes@pdmfc.com, joao.luis@pdmfc.com, luis.campos@pdmfc.com

June 2000

Abstract

Tea is a high level scripting language for the Java environment. It has built-in support for all major programming paradigms, namely procedural, object oriented and functional. Its major strengths reside in its consistency, simplicity, easy extensibility and easy integration into any Java environment. These advantages are in addition to the intrinsic advantages of being a scripting language, these being rapid development turnaround, fast prototyping, glueing logic for components.

1 Introduction

Tea is a high level scripting language for the Java environment. Its initial development was primarily motivated by the lack of scripting languages for the Java environment. Tea also aims at overcoming some of the limitations of current scripting languages. In fact this goal was the main reason for the development of a new language, instead of merely implementing an interpreter under the Java environment for an existing language.

When work first began on the Tea programming language in mid 1997 there were no interpreters for scripting languages that could be used from within a pure Java environment. Three years later there are now at least two mainstream scripting languages with interpreters written in Java, these being Tcl [2] and Python [4]. Nevertheless the reasons that drove the initial development of the Tea language still hold. Moreover Tea has more than met the original expectations, as will be made clear when its uses in real world projects are presented later in this paper.

The purpose of this paper is twofold. Firstly it gives an overview of the Tea language and its features. Secondly it describes its advantages over competing products and lists some real world projects where Tea has been used as a core component.

The rest of this paper is organized as follows. Section 2 gives an overview of the Tea programming language. Section 3 addresses some of the trade-offs that arose during the design of the language. Section 4 gives a short description of the main functionalities of the Tea core libraries. Section 5 presents the Tea Object System which allows for object oriented programming. Section 6 discusses some of the advantages of the Tea language. Section 7 mentions some real world projects where Tea has been used. Section 8 makes reference to some current work being done on Tea. Finally, section 9 concludes the paper.

2 Tea Overview

Tea is a functional language at its heart. In a superficial contact its syntax seems to resemble the syntax of Tcl. The semantics are similar to those of Scheme. It has real closures and functions as objects. However these similarities are deceiveing since Tea is quite different from these two languages. This will become quite clear in the following sections.

A Tea program consists of a sequence of statements. A statement is a function invocation, with a given set of actual arguments. In a Tea program statements are separated by end-of-line characters or by semi-colons. A statement is a sequence of words. The first word identifies the function that will be called. The remaining words are the actual arguments passed in the function call. Words are separated by white space. A word can be an object literal, a variable substitution construct or a command substitution construct.

2.1 Basic Tea Types

There is a small set of pre-existing object types. These are as follows: symbols, strings, numeric objects, boolean values, lists, code blocks and functions. In addition there is another type of object for which there is a single instance: the null object. All the entities manipulated in a Tea program are objects of these types. For some of these types it is possible to have literal representations of an instance in a program (e.g strings, numeric values). For other types instances are only created by functions or exist in pre-initialized variables. Objects of other types in a Tea program can also exist. These are created and manipulated by specialized functions.

- Symbols. In a Tea program a symbol literal appears as a sequence of characters, with no white space and not starting with a digit nor with any of the following characters: `"`, `[`, {, `$`, `(`. A symbol is mainly used to identify variables.
- Strings. A string literal is any sequence of characters enclosed in quotes. A quote character inside a string is represented as a backslash character followed by a quote character. A backslash character inside a string literal is represented as two consecutive backslash characters.
- Integer and floating point numbers.
- Boolean values. The interpreter automatically creates two variables, named "`true`" and "`false`" containing the true and false boolean values, respectively.
- Lists. A list literal appears in a program as a sequence of words enclosed in parenthesis. Words in a list are separated by white space or end-of-line characters.
- Code blocks. A code block is a sequence of statements enclosed in brackets.
- Functions. Functions are first class objects. There is no function literals. There are, however, pre-defined functions that return function objects (e.g. `lambda`).
- The null object. Although there is no literal for the null object there is a variable, named "`null`", that is pre-initialized with a reference to the null object.

2.2 Variables

Variables are an intrinsic part of the language. Variables do not hold actual objects. Instead they are used as bins that hold references to objects. You can have two or more variables that reference the same object.

Access to a variable contents is achieved through a syntax construct, namely variable substitution. Variable substitution is done by preceding a symbol with a dollar sign. Whenever the interpreter encounters a word of this type the word evaluates to the variable contents.

2.3 Command Substitution

Command substitution occurs whenever there is a sequence of commands enclosed in square brackets. For example, the following command

```
set! i [+ $i 1]
```

could be used to increment the value of variable "`i`" by one. This explanation is an oversimplification. What actually happens is this: after the command is executed variable "`i`" will hold a reference to an integer object representing the value obtained by adding one to the value represented by the integer object whose reference was previously held in the "`i`" variable.

As an example, here is a factorial function.

```
define factorial ( n ) {
    if { < $n 2 } 1 { * $n [factorial [- $n 1] }
}
```

2.4 The Tea Interpreter

When a Tea interpreter executes a Tea program, two steps are performed internally. First the source file is parsed and an efficient representation of the program is compiled and stored in internal data structures. Only then does the actual execution of the program begins. This way, source files are read and parsed only once.

There are no mechanisms for explicitly allocating and releasing memory. Instead, the Tea runtime environment relies on the underlying Java Virtual Machine garbage collecting facilities to release unused memory space.

Whenever a runtime error occurs, such as calling a function with an incorrect number of arguments, an error is thrown. If the error is not catched it will unwind the whole calling stack of Tea functions until the program terminates. A message with a dump of the call stack is then printed to the process standard error stream. It is possible in a Tea program to catch runtime errors by using the "`catch`" function.

3 Design Considerations

The design of the Tea language was driven by three major goals, namely:

- Syntax simplicity.
- Expressiveness.
- Extensibility.

Having a simple syntax as a design goal ensured the ease of learning for every newcomer to the language. As a further advantage the implementation of the language parser was also made simpler. The language intrinsic simplicity draws on Scheme and Tcl. In fact Tea programs often look like Scheme programs with the Tcl

syntax. Nevertheless Tea is very different from these two languages. For instance, in Tcl everything is a string while in Tea every object has a certain type. As for comparing with Scheme, there is no macro mechanism like in Scheme. There is no need for one, actually. In Tea you explicitly perform variable evaluation (through variable substitution) and code block execution (through command substitution).

The language expressiveness derives from its functional roots. Tea is a functional language at its core. It has all the features expected from a true functional language, such as functions as objects and closures. Although a functional language Tea also supports two other major programming paradigms, namely, procedural programming and object oriented programming. Support for these other programming paradigms was achieved by deploying a minimum set of functions with all the required features. Functions like `if`, `while`, `foreach` allow for conditional execution and cycles. As for object oriented programming, it is supported through functions like `class`, `method`, `new`. Section 5 gives an overview of the object oriented features of Tea.

Tea was also designed to is be easily extensible. Extensions take the form of new sets of functions and classes that provide additional functionalities not present in the core library. Since the Tea interpreter is written in Java, new extensions may and have been rapidly implemented on top of one of the varied standard Java APIs. As an example, the TDBC module, which is now part of the Tea core, is implemented with the JDBC API.

4 Tea Core Libraries

The simplicity of Tea as a language is complemented by a rich set of functions. This set of functions is an integral part of the Tea language itself. These functions provide all the niceties that are expected from a general purpose programming language. The following list enumerates sets of functions and classes with related functionalities.

- IO – Functions and classes for file manipulation and I/O.
- Lang – Tea core functions.
- Lists – List processing.
- Math – Numeric functions.
- Regexp – Functions involving regular expressions.
- String – Manipulation of strings.
- TDBC – Tea database connectivity. Set of classes for accessing relational databases.
- TOS – Tea object system. Set of functions providing object oriented capabilities to the Tea language. See section 5 for additional information.
- Util – Utility classes and functions (vectors, hash tables).
- XML – Classes and functions for processing of XML files.

This rather large set of functions would seem to imply that every Tea program would be bloated with this much code. This is naturally not the case. The Tea runtime system makes use of the dynamic loading facilities of the Java language. The Java code that implements a Tea function is only loaded into the Java Virtual Machine, on top of which the Tea interpreter runs, the first time the Tea function

```
class Rectangle (
    _width
    _height
)

method Rectangle constructor ( w h ) {
    set! _width $w
    set! _height $h
}

method Rectangle getArea () {
    * $_width $_height
}
```

Figure 1: The definition of a `Rectangle` class.

is called. This way only the code that is actually needed is ever loaded into the Java Virtual Machine, leading to a very efficient memory usage. For functions and classes implemented in Tea there is also a similar auto-loading mechanism. Using this mechanism the source file where a function or class is defined is executed only the first time the call to the function or instantiation of the class is encountered.

5 Tea Object System

Included in the Tea language are features to support object oriented programming. These features are not part of the language itself. They are, instead, implemented through a set of functions providing all the needed functionalities.

An object is data with a set of methods that operate on its data. An object is an instance of a class. The concept of class is not built into the language. Instead, a class in the Tea Object System is represented by a Tea object. To create a new TOS class the `class` function is used. When a new class is created its members are also specified. In figure 1 we show an example of a class `Rectangle` definition.

After a class object is created it is possible to associate methods with that class. In the Tea Object System all methods are public. The `method` function is used to associate a new method to an existing class. When the code block representing the body of a method is executed it has access to variables that are aliases to the object's members. In addition to members variables two other special variables are automatically created and initialized inside the method body. The `this` variable contains a reference to the object for which the method was called. It is used to call other methods from within the method body. The other special variable is `super`. It also represents the object itself, but it is used to invoke methods from the base class.

To create an instance of a TOS class the `new` function is used. This function receives as the first argument the symbol identifying the class. Any remaining arguments directly passed to the class constructor. A TOS object is also a function. It is through this function that the object methods are invoked. It receives as first argument a symbol identifying the method to call. Any additional arguments are passed as arguments to the method. As an example, the following code snippet creates an instance of the `Rectangle` class (see figure 1) and displays its area on the standard output.

```
define rect [new Rectangle 3 4]
echo [$rect getArea]
```

```
class Square Rectangle (
)

method Square constructor ( size ) {
    $super constructor $size $size
}
```

Figure 2: Definition of a `Square` class, derived from `Rectangle`.

The TOS also suports class inheritance. A new TOS class can be defined as being derived from a previously defined base class. By design, only single inheritance is possible. The derived class inherits the members and methods of its base class. Figure 2 shows an example of a new class `Square` derived from `Rectangle`. All the members of a class are private. That means they can be accessed only from within the methods of that particular class. They are not accessible in methods of derived classes.

6 Advantages of Tea

As a scripting language Tea shares all the advantages over system programming languages that are intrisinc to scripting languages [1]. These advantages can be summarized in the three points enumerated bellow.

- Rapid turnaround during development. Because scripting languages are interpreted there is no compile-link phase. Changes to programs can be made and tested on the fly.
- Fast prototyping. Scripting languages are very high level programming languages. Programs tipically require much less code and development time [2] when compared to their equivalents implemented in a system programming language.
- Glue between components. One of the major uses of scripting languages is as glue languages to assemble disparate components. The components are written in a system programming language (C, C++, Java) and have very specific purposes. The typeless approach common to scripting languages makes it much simpler to combine components together. Code reuse is thus naturally propitiated in a scripting language environment.

Besides all these advantages inherent to scripting languages, Tea has three major advantages that are specific to this language.

- Simple syntax and consistency.
- Easy extensibility. By design it is easy to add new functions and classes implemented in Java to a Tea interpreter. Because the current implementation is based in Java it is only natural to draw from the large set of standard Java APIs to augment the existing functionalities.
- Natural integration into any Java environment. The major strength of the Tea language lies in its Java foundations. A Tea interpreter can be used embeded in a Java application or framework to provide scripting functionalities. In standalone programs, Tea can be used as the glueing logic to seamlessly integrate separate Java modules.

7 Real World Projects in Tea

The Tea language has been used in real world projects ever since its inception in mid 1997. One of the major uses of Tea has been as the programming language for an application server totally based on Java, the I⋆Tea application server.

Several large projects have used this application server as the base for web applications. Included among these are three internet homebanking applications, for three major banks in Portugal (BPSM[1], BTA[2] and CPP[3]), and one on-line brokerage application[4] (about to enter production phase at the time of this writing). The Tea scripting language was used to code all the logic for the presentation layer of the web applications.

In one other large project (involving over a million lines of code) for the major ISP in Portugal (Telepac), Tea was also used as the programming language for an integrated ISP management system. This is a business critical system, working 24/7. This system includes extensive CRM components (e.g. customer self-service and help-desk applications) and billing modules.

Tea has, therefore, already proven itself in the field, having been used as a core component in several mission critical bussines projects.

8 Future Directions

Tea has been continuously evolving since its creation. Although the syntax definition and the core set of functions has been stable from the begining new functions and classes are added to the package with each new release.

At the present time two new modules are being actively developed. The first of these modules will enable the creation of Tea applications with a GUI. The other module involves adding new functions and classes to support networking operations.

The new module for creating GUI applications written in Tea is on its final stage of design. The implementation will be based on the Swing widget set [5]. One major design requirement is integrated support for arbitrary JavaBeans [6]. Programmers will then be able to assemble different GUI widgets that are JavaBeans from, possibly, distinct vendors to create the final application GUI using Tea code as glueing logic between these components. The Swing usual components (buttons, labels, panes, etc) are also considered JavaBeans. This means they will be used just as any other JavaBeans components. The proposed Tea API for GUI applications will give a unified treatment to all the GUI components.

The networking module will provide functions and classes for socket programming.

9 Conclusions

Tea is a new scripting language designed specifically for the Java environment. It includes natural support for all major programming paradigms (procedural, object oriented and functional). Its major strengths reside in its consistency, simplicity, easy extensibility and easy integration into any Java environment. These advantages are in addition to the inerent advantages of being a scripting language. The Tea language provides a powerful addition to the tools that programmers should have at their disposal.

[1] `https://www.bpsm.pt/general/EntradaBpsm.html`
[2] `https://www.bta.pt/homebank/general/EntradaBta.html`
[3] `https://www.cpp.pt/homebank/general/EntradaCpp.html`
[4] `http://www.ljcarregosa.pt`

The experiences with using Tea have been extremely positive. It has already been in use for over three years in mission critical projects. Its most proeminent uses being in the development and deployment of web applications.

Aknowledgements

The authors would like to thank João Costa and Paulo Correia who reviewed early drafts of the manuscript and gave many helpful comments.

References

[1] J. Ousterhout, *Scripting: Higher Level Programming for the 21st Century*, IEEE Computer Magazine, March 1998.

[2] J. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[3] K. Arnold, J. Gosling, D. Holmes, *The Java Programming Language*, 1996.

[4] M. Lutz, *Programming Python*, O'Reilly, 1996.

[5] D. Geary, *Graphic Java 2, Volume 2, Swing*, Prentice-Hall, 1999.

[6] R. Englander, M. Loukides, *Developing Java Beans*, 1997.