Tche - a Visual Environment for the Lua language

ANDRÉ FILIPE LESSA CARREGAL
ROBERTO IERUSALIMSCHY

Departamento de Informática
PUC-Rio
Rua Marquês de São Vicente, 255
22453-900 Rio de Janeiro, RJ, Brasil
{carregal, roberto}@icad.puc-rio.br

Abstract: The Tche (Tiny Canvas-Hosted Environment) system is a visual environment that supports the direct manipulation of visual objects written in the Lua language. Tche was designed to be simple, portable and flexible, being also highly user customizable. Tche uses a workspace where the visual objects can be manipulated, modified and browsed. The system allows the use of text Lua commands, offering real-time interactions not available in conventional systems. We introduce other visual environments such as Visual Basic, ACE, Smalltalk, Oberon, SELF and Tcl/Tk, and briefly discuss the solutions and characteristics of each system in the direct manipulation of visual objects, ease of use and portability.

Keywords: Visual Programming, User Interface, Abstract Data Views, Active Objects, Visual Objects.

1. Introduction

The advent of new programming styles usually brings up new software development tools. The need to combine prototyping and rapid application development, makes necessary the use of specific tools, to achieve flexibility and ease of use. These tools normally use the concept of direct manipulation of objects. Those can be considered *visual objects* due to the existence of a graphical representation corresponding to each object.

Software tools with such characteristics are called visual development environments. Some of these tools allow visual objects to have their attributes and behavior modified in run time (dynamic modification). This feature enables the investigation of new possibilities and combinations, stimulating an exploratory programming style.

This work describes the Tche (Tiny Canvas-Hosted Environment) system. Tche is a visual development environment that supports the direct manipulation of visual objects described in Lua. Lua [FIC94] is a programming language with structured language characteristics and object orientation, based in a portable interpreted environment. Lua can be used as a configuration language, using its data description capabilities, as much as a programming language embedded in a development system.

Tche is simple to use and requires few machine resources and a little learning time. Nevertheless, the system is also flexible enough to allow a high degree of user configuration. The system also offers easy portability, minimizing the utilization costs in heterogeneous environments.

In the next section we introduce some related visual environments. Then we describe the Lua programming language and the Tche System. Following this, we conclude with some suggestions for future works.

2. Related work

This section introduces other visual environments such as Visual Basic, ACE, Smalltalk, Oberon, SELF and Tcl/Tk, and briefly discuss the solutions and characteristics of each system in the direct manipulation of visual objects, ease of use and portability.

The Visual Basic system[MS92] offers a visual development and programming environment allowing simple prototyping and complete program creation using a Basic language variation. The program interface creation is done with direct manipulation of interface elements. Each element has a set of properties and a programmed behavior. The system can use the built-in elements and/or C created elements. The last ones allow different functionalities and features to be added to the system.

Visual Basic offers no run time object direct manipulation, unless the developer explicitly implements this feature in Basic code. Although Visual Basic builds stand-alone executable application, those are not portable running only in the Microsoft Windows plataform.

The ACE (*Application Construction Environment*) system [JNZM93] uses the composition of high-level

semantic components to develop applications. The ACE system uses a high-level component manipulation language based on the spreadsheet formula definition. The language is used to configure and modify the application components. In traditional systems the programmer uses low-level abstraction resources to specify what the application does and how it interacts with the user. In opposition to this, a spreadsheet offers functionalities and interface elements that are configured by the user to develop an application.

Smalltalk [Gold84a] consists in an object-oriented programming language and a group of integrated tools that manipulates the various components of the Smalltalk system. The system display is divided in rectangular frames containing text and/or pictures. Frames can be moved, resized, collapsed or removed.

Smalltalk uses the concept of *image* to store and recover its configuration. The user can at any time save the system image so the modifications become part of the system. Smalltalk offers tools such as text editors, class browsers and an interactive debugger. The class browser allows the visualization and modification of the system methods and attribute definitions. New classes and methods can also be added as needed.

Oberon [Wirt92] is derived from Modula-2 and was created to develop operating systems. Oberon differs from Modula-2 mainly by offering type extension support, essential to the system expansion. The Oberon environment offers advanced resources for the editing and formatting of text and pictures. The system is extremely economic under the perspective of resource utilization. The display management is based in frames, positioned by the system using specific space-saving algorithms. The system uses frame tiling instead of the more common overlapping method. The system is available for many platforms and porting Oberon programs is very easy, involving only recompilation.

SELF [AB93] is an object oriented language, developed to facilitate the exploratory programming. The language includes dynamic typing and garbage collection. SELF does not use the concept of classes or variables, adopting instead the prototyping of objects. The object attributes (*slots*) are accessed sending messages to *self*.

The SELF environment has a high-level abstraction of the system objects, allowing the direct manipulation of object properties. The environment uses sophisticated interface methods, including cartoon animation techniques. That sophistication, however, makes SELF a resource-demanding system. Another negative points are the lack of portability and the deep dependence to the SUN platform.

Tcl (*tool command language*) [Oust94] is a script language developed for application control and extension. Tcl uses a C library interpreter that can be embedded in the application. The use of the same language across a set of applications allows the exchange of scripts by the applications. This communication mechanism is more powerful and flexible than static libraries solutions like Microsoft OLE and SUN ToolTalk.

The Tk [Oust94] toolkit is a Tcl extension that offers X11 user interface building commands. Tk allows the creation of interfaces using Tcl scripts instead of C code. The toolkit uses X11 to implement a widget set based on the Motif look and feel. Each widget belongs to a class that specifies the appearance and behavior of the widget. The behavior is determined by a Tcl script binded to the widget.

3. The Lua language

Lua [FIC94] is a dynamic typed extension language with traditional structured and object-oriented characteristics. Lua is implemented as a C library and does not have the concept of a main program, being used always within a host program. Lua reflexivity mechanisms allow the creation of new code chunks and the association of this code to the system, facilitating the implementation of persistent objects.

Before executing a piece of code, Lua compiles its commands and functions and then executes the commands sequentially. Functions are values in Lua and can be stored in variables, used as parameters and returned from other functions. The *table* type implements associative arrays that can be indexed by any value, being used not only as conventional arrays but also as lists, symbol tables, records, etc.

The table constructor mechanism allows the implementation of very sophisticated data structures, promoting the use of Lua as a powerful configuration language. The table constructors can create empty tables or tables with initialized values. Constructors can call a Lua function, a feature that can be used to many purposes, such as defining default values for the created table.

Lua has a powerful semantic extension mechanism called *fallbacks*. This mehanism allows a programmer to define functions to be called when Lua does not know how to handle an error situation. A common use for fallbacks is the implementation of an inheritance

mechanism. More details about the language and its API, including code examples and the integration of Lua and C can be found in the language reference manual [IFC95].

4. The Tche System

The Tche (Tiny Canvas-Hosted Environment) system [Carr95] is a visual environment that supports the direct manipulation of visual objects written in the Lua language. Tche was designed to be simple, portable and flexible, being also highly user customizable. Tche uses a workspace where the visual objects can be manipulated, modified and browsed. The system allows the use of text Lua commands, offering real-time interactions not available in conventional systems.

The acronym choice intends to convey the philosophy of an economic system easily portable to most graphic platforms. Tche uses some concepts of the systems presented before, including some ideas not present in these systems.

Being an event-driven environment, Tche does not have a main program. Instead, the system consists of different objects that interacts with themselves and with the user. Like some similar systems, Tche can use timers to generate events in regular time intervals. The Tche event handling architecture permits the creation of new event types and the association of these to the system. As an example of this mechanism, events can be added to the system using RPC (Remote Procedure Calls), allowing the remote system manipulation.

As Smalltalk, Tche uses a system image concept to store the visual objects state. However, Smalltalk uses a complete system state recording. Instead of this, the Tche system image consists of a Lua text file describing the system objects. Using a configuration file like this allows the transport of Tche applications within different platforms, helping the interface and prototype development.

Tche can be used in three levels: in the top level, application users operate the system through its interface; in a middle level, Lua developers modify the system using Lua code; and in the lower level, C programmers add new capabilities, such as events, to the system.

From the application user perspective, the Tche environment is composed by a command window and one or more canvas, where visual objects are placed. The command window allows the typing of any Lua command to be executed. The user can also activate browsers to navigate and modify the visual objects attributes (Figure 1). The browsers let the user edit simple attributes and activate new browsers to access table type fields. Simple attributes (numbers and strings) can be modified right into the browser, while table and function attributes are just highlighted indications.

Tche uses some SELF concepts to show visual objects. Objects are solid and move like filled blocks, instead of border frames. The appearance of new objects in the screen, including menus, uses animation techniques to simulate the *breeding* of the object. The simple techniques used show that, the seemingly little efforts in the interface interaction and visualization, make a real difference in the system user feedback.

Unlike most of the other systems, Tche does not force an object to be positioned in front of the others while moving. A moving object keeps its relative position, passing over objects that are in lower positions and under superior ones. To enhance the movement illusion the system uses a double buffering technique like the Tk toolkit [Oust94]. Notice that this moving metaphor needs fast bit map operations (BitBlt) to run smoothly.

To a Lua developer the Tche system offers a hierarchy of objects defined as Lua tables. Each object has attributes that can be simple values, tables or functions. The use of functions as table attributes can be interpreted as an association of methods to objects. The system hierarchy offers objects corresponding to windows, lists, timers and the visual object manipulation subsystem.

Tche uses an inheritance mechanism to allow an object to access attributes of hierarchically higher objects. If an object does not have a certain attribute, the system use an attribute named *parent* or *godparent*[1] continue recursively the search. This mechanism permits the use of multiple inheritance in the hierarchy.

Tche uses *prototyping* to create new objects. A prototype is a Lua object with attributes to be used by its heirs. An heir is a Lua object which *parent* or *godparent* attributes are references to the prototype or to an heir of the prototype. Methods and attributes defined for a prototype can be redefined in its heirs, without changing the prototype original values. This mechanism is also used in SELF and parallels the traditional class-based inheritance definition.

Each visual object has a set of associated methods named *callbacks*. Each callback is triggered by the system when events occur over the object. This works like the *binding* of Tcl/Tk widgets, allowing the redefinition of the object behavior through the modification of the

[1] the name *godparent* comes from the idea of "parent substitute".

corresponding callback. Tche defines callbacks for mouse movements, keyboard events and for changes in the canvas size and position.

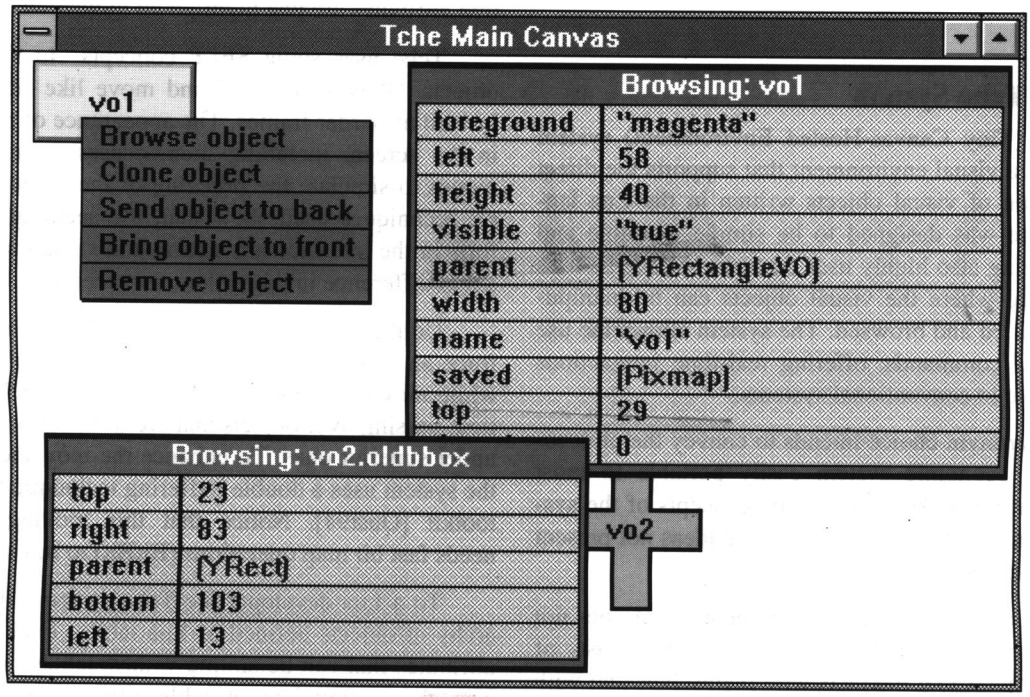

Figure 1 - Visual objects, menus and browsers

The use of Lua allows the modification of the system at any time. The user can redefine functions in run time and assign these functions to an object attribute or callback. The modification of Lua code can be done in the command window or with the help of an external editor.

The handling of the visual objects and their callbacks is done through the UAI framework. UAI [BCCI94] is a C++ class framework that allows the development of graphic-interactive programs with visual objects support. UAI offers graphics operations and a group of classes responsible for the manipulation of visual objects (*VO*). The Tche system uses a Lua implementation of the UAI framework. However, instead of using a class hierarchy, Tche implements UAI as a group of prototypes corresponding to the original C++ classes. This does not differ from the UAI defined concepts.

The VO concept in UAI corresponds to *Abstract Data Views* [CILS93], modeling active objects. Active objects have a behavior determined by their classes, reacting to events triggered by user interaction or by other objects. Besides handling events, active objects can show themselves in the screen using a drawing method.

To the UAI/Lua framework a *pen* is any object that offers drawing methods. It is possible, thus, to create a virtual pen, that draws directly in memory, not in the screen. A virtual pen simplifies the use of transparent double buffering techniques to draw the VOs. A VO does not need to know where it is drawing itself, only how it should draw itself. The system handles every needed optimization and animation.

Tche allows the creation of visual objects that heir to sophisticated prototypes. Some prototype examples are the *draggable* and rectangular filled objects. A draggable object can be moved through the canvas with the use of drag and drop operations. Rectangular filled objects can have its bitmap stored in memory to speed up the redrawing process when moving across the screen. Although seemingly simple, this optimization benefits most of the traditional interface elements.

To store a system image, Tche uses the Lua reflexivity to describe the data structures corresponding to the system objects. In the rectangular VOs (*YRectangle*), the object image is the direct description of the rectangle attributes. The Tche image file consists in a series of object constructors. To restore the system image, Tche uses Lua to open, compile

and execute this file. The constructor used for the image file is the same one that the user can call to insert a new VO in the system using the command window.

As an example of the image file format, to save the image of a system as the one presented in Figure 2, Tche creates an image file with a format similar to the show in Figure 3.

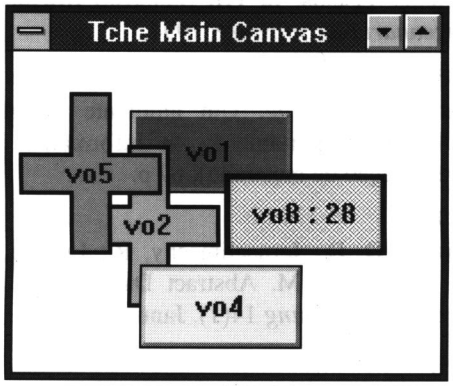

Figure 2 - Tche example

Finally, from a lower level perspective, instead of making use of a traditional language (C or C++), we decided to use Lua as the main language, and C as the language for the system kernel. Lua is interpreted and easily portable, conferring every flexibility needed. The choice of C over C++ was due to the higher portability achieved with C.

```
RectangleVO {
     left = 58,
     top = 29,
     width = 80,
     height = 40,
     foreground = 'magenta',
     name = 'vo1'
}

CrossVO {
     left = 32,
     top = 46,
     width = 70,
     height = 80,
     foreground = 'cyan',
     name = 'vo2'
}
```

Figure 3 - Image file example

Like Oberon, Tche uses simple graphical primitives to draw and show text. The straightforwardness of this set of primitives simplifies its implementation and, therefore, the system porting. The higher level operations are coded in Lua, granting to any platform that implements the primitives, the full system potential. Notice that Lua is interpreted but does not impose any performance penalty on the Tche system. The system performance is directly bound to the target machine graphics capabilities.

5. Conclusions

This work introduce the Tche system and some related visual environments such as Visual Basic, ACE, Smalltalk, Oberon, SELF and Tcl/Tk. These environments contributed in differents ways to the Tche system.

Visual Basic showed up as a very sophisticated development environment, but did not offer some needed features as the dynamic code manipulation and code portability.

The ACE system contributed with the concept of using a common language to define the behavior and attributes of objects. ACE presented, however, a inherent structural complexity.

Smalltalk was an excellent example of the utilization of browsers, exhibiting, however, problems of consistency and portability due to the different versions of the system.

Oberon showed us that a system could be based in simple basic principles and graphic primitives and still keep up with the performance needs.

SELF contributed with fundamental concepts to the Tche system. Some examples are: the concept of prototyping instead of class-based instantiation of objects, the linked objects hierarchy and, last but not least, the use of animation techniques and object solidity to enhance the user interactive feedback. The sophistication of the SELF system, although, revealed too much for the intended purposes.

As the last system analyzed, Tcl/Tk demonstrated an excellent integration of an embedded language and a visual objects toolkit. The concept of *binding* in Tcl/Tk is also very similar to the *callback* mechanism used in Tche.

Besides offering the concepts shared with these systems, Tche added some other notions and capabilities. The event-driven architecture empowers the creation and aggregation of new dispatchers to the system. As an example, during the development of the Tche system, a related project implemented ex-

ternal events. These events are triggered and manipulated using encapsulated RPCs (*Remote Procedure Calls*), allowing the remote manipulation of the Tche system.

Tche fulfilled its proposed purposes, demonstrating the Lua potential to implement complete systems. From a performance perspective, Tche testified that Lua is up to the task. The system performance is much more tightly linked to the graphical processing capabilities (mainly *BitBlt* operations) of the target-platform.

Following the development of the Tche system, some possible additions become attainable. One of them is the utilization of *function persistence*. Persistent functions can be modified in run time and included in the system image, being stored and recovered with the objects. From a manipulation perspective, another addition is the possibility to group objects, allowing composition of simple objects in more complex ones. The *connector* concept is also a very important addition to the system. With connectors the user would be able to bind semantic relationships to links between objects, a concept similar to the used by constraint programming systems.

6. References

[AB93]
Agesen, O.; Bak, A. The SELF 3.0 Programmer's Reference Manual. Sun Microsystems, 1993.

[BCCI94]
Borges, R.; Cassino, C.; Cerqueira, R.; Ierusalimschy, R. UAI - Um framework para suporte a Objetos Visuais. In *VIII Simpósio Brasileiro de Engenharia de Software*, 1994.

[Carr95]
Carregal, A. *Tche*: Um ambiente visual Lua. PUC-Rio, Departamento de informática. 1995. (Dissertação de mestrado), 67 p.

[CILS93]
Cowan, D. D.; Ierusalimschy, R.; Lucena, C. J. P.; Stepien, T. M. Abstract Data Views. *Structured Programming* 14(1), January 1993, p. 1-13.

[FIC94]
Figueiredo, L. H.; Ierusalimschy, R.; Celes F., W. The Design and Implementation of a Language for Extending Applications. In XXI Semish, 1994, p. 273-284.

[Gold84a]
Goldberg, A. *Smalltalk - The Interactive Programming Environment*. Addison-Wesley, 1984. 516 p.

[IFC95]
Ierusalimschy, R.; Figueiredo, L. H.; Celes F., W. *Reference Manual of the Programming Language Lua* version 2.1. PUC-Rio, 1995. 24 p.

[JNZM93]
Johnson, J. A.; Nardi, B. A.; Zarmer, C.; Miller, J. R. ACE: Building Interactive Graphical Applications. *Communications of the ACM* 36(4), April 1993. p. 41-54.

[MS92]
Visual Basic Programmer's Guide. Microsoft, 1992.

[Oust94]
Ousterhout, J. K. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994. 458 p.

[Wirt92]
Wirth, N.; Gutknecht, J. Project Oberon - The Design of an Operating System and Compiler. Addison-Wesley, 1992. 547 p.