Writing Object-oriented Tcl-based Systems using Objectify

Wayne A. Christopher*

University of California, Berkeley

Abstract

This paper describes Objectify, a tool that facilitates the integration of C++ classes into Tcl-based systems. It uses an object model similar to that of Tk, and makes it possible for the programmer to annotate his classes with information about which member functions should be exported as Tcl "methods", and which data members should be accessible from Tcl via "configure" commands. Objectify automatically generates glue code to interface between Tcl and C++, and also creates documentation files. This approach is fairly light-weight and straightforward, and automates the tedious and error-prone task of maintaining Tcl command interfaces to lower-level object implementations.

1 Introduction

Tcl provides a very flexible and simple interface between code written in Tcl and C or C++ code. One can create commands in an interpreter using `Tcl_CreateCommand`, and provide for the command to be called with an arbitrary piece of "client data" as one of its arguments. No restrictions are imposed on when new commands can be created or what the client data can consist of.

In many cases, however, one wants to impose some structure on this scheme in order to support more disciplined styles of programming. An example is "object-oriented Tcl", which is described in Chapter 3 of the as yet unpublished Tk/Tcl book [1]. The major existing instance of this style is the Tk toolkit – widget objects are created using a *creation* command, such as `entry .ent -bg green`, methods can be called on them, for example `.ent get`, their instance variables can be examined and changed using the `configure` method, and they can be deleted using the `destroy` command. An object is made available to the Tcl level by registering its name with the interpreter as a command, using the address of the data structure that stores information about the widget as the client data argument.

*Computer Science Division, University of California, Berkeley, CA 94720. Email: `faustus@cs.berkeley.edu`. Part of this work was done at Pacific Marketing and Communication, Berkeley CA.

The system described in this paper, called Objectify, provides a simple and convenient way to define objects in the style of Tk widgets, with slots and methods. For each Tcl object there is one C++ object, and methods and slots are implemented via member functions and instance variables. The major requirements that Objectify imposes on the programmer are that he be free to modify the relevant class declarations, and that he not mind using a few unsightly macros in these declarations.

The Objectify program does two things. First, it reads the declarations for classes that are to be made into Tcl object types and automatically generates most of the "glue" code to interface between the Tcl and the C++ worlds. Second, it uses the information contained in these classes, which includes usage annotations, to generate a human-readable summary that documents the class and its methods and slots. It tries to facilitate what Knuth calls "literate programming" [2], by making it easier to maintain code and documentation in parallel and to keep them synchronized.

Recently there have been a number of suggestions for making the integration of Tcl with lower-level object systems easier. The tclOBST system [3] merges Tcl with a persistent schema-based object system. In [4], a technique is described for automatically linking C data structures with Tcl variables, which provides fine-grained access capabilities. Other researchers have developed or designed systems that more directly address the problems that Objectify deals with, but to my knowledge none of these have been published or released yet.

2 Using the system

Objectify, which is a Tcl program, operates on a file by file basis. It takes one header file, for example, `test.h`, and produces C++ and documentation files called `test_objects.cc` and `test_objects.doc`, respectively. The header file contains one or more specially annotated class declarations. An example is shown in Figure 1.

The class declaration contains a number of lines that begin with `TCL_`. The meanings of these lines are as follows.

```
class TCL_OBJECT("thingie", Thingie, object, "This command creates a new thingie.") {
   public:
      Thingie(Tcl_Interp* interp);
      ~Thingie();
      TCL_SLOT(BOOLEAN, int, useful_flag, "-useful", "useful", "Useful",
           "1", 0, NULL, "<boolean>", "This flag determines whether the thingie is useful.");
      TCL_METHOD("apply", ApplyCmd, 0, 2, "[ times ] [ intensity ]",
           "Apply the thingie the specified number of times with the specified intensity.");
      TCL_METHOD("configure", ConfigureCmd, 0, -1, "options", "Configure the thingie.");
      TCL_METHOD("delete", DeleteCmd, 0, 0, "", "Delete the thingie.");
      int AfterConfigure(Tcl_Interp* interp); // Required member function.
};
```

Figure 1: An example of a Tcl C++ class definition.

- `TCL_OBJECT`: This defines the name of the class and the Tcl command that can be used to create instances of the class. The type may be either "object" or "widget". In the latter case, the system will generate some extra code to create windows and do Tk-related bookkeeping. A documentation string for the class is also included here.

- `TCL_SLOT`: This specifies a slot, or data member, within the object, which can be accessed via the `configure` method. The arguments describe both the C++ type and name, and the name by which it can be accessed from Tcl. (In general, this name will begin with a "-".) There is also a form, `TCL_SLOT1`, that omits the options database information, which can be used when the type is "object" rather than "widget". Documentation strings can be given that describe the type of the slot and what it is used for.

- `TCL_METHOD`: This specifies the Tcl name and the C++ name of the member function, along with information on the number of arguments allowed. The actual member declaration is always of the form

  ```
  int MethodCmd(Tcl_Interp* interp,
                int argc, char** argv);
  ```

  The argument number constraints apply to the value of `argc`, and a maximum value of -1 indicates no upper limit on the number of arguments. Documentation strings must be given that describe the arguments and the purpose of the method.

When the header file is compiled as C++ code, these lines are macro-substituted so as to provide the appropriate declarations. The macro definitions are contained in the file `objectify.h`, which must be included in this header file. Since these definitions expand in a fairly straightforward way, one can easily define derived classes, virtual methods, and inline method definitions. The Objectify program only looks at what is in the argument lists of the `TCL_` entries.

The `argc` and `argv` that are passed to the functions implementing the method are adjusted so that `argv[0]` points to the first argument after the method name. No attempt at parsing the values of the arguments and calling the function with C++ types rather than strings was made, in the interests of keeping the interface as simple as possible.

When an object is created, its name is either specified in the command, if it is a widget, or is automatically generated, if it is a regular object. In either case it is returned as the result of the creation command. The automatically generated names are of the form *name_address*, where *name* is the Tcl object name and *address* is the hex address of the object. While the use of absolute addresses for object handles is frowned upon in the Tcl community, I feel that the use of the object name in the handle allows adequate error checking. The problem of dangling references remains, however, and the use of lookup-table based handles would be a useful enhancement to Objectify.

If the object type is "object", then a constructor must be provided that takes a `Tcl_Interp *` as an argument. If it is "widget", then one must be provided that takes a `Tcl_Interp *` and a `Tk_Window`. It will be called after the window is created, and will probably want to store this data somewhere. There is no way for the constructor to indicate failure to the creation routine, but this can be done in the `AfterConfigure` routine, described below.

Alternate forms of the `TCL_` entries exist, `TCL_METHOD_PARENT`, `TCL_SLOT_PARENT`, and `TCL_SLOT1_PARENT`, which do not macro-expand into member definitions and can be used when the members are inherited from a parent class.

A few special methods can be declared, whose implementations are automatically created by Objectify. If they are omitted, no code will be generated for them.

- `configure`: This method works like the Tk configure operation. If it isn't declared, the slots can be initialized in the creation command but not examined or modified from Tcl.
- `delete`: This method removes the object command from the interpreter and applies the C++ `delete` operator to the object. This approach is a bit different from the Tk mechanism, which uses a single `destroy` command to delete all widget types. The use of a deletion method is cleaner from an implementation standpoint, and I feel it is more in keeping with the object-oriented philosophy.
- `help`: This method returns the documentation strings provided in the class declaration to the user. Without arguments, it returns the class documentation and a list of the methods and slots, and when given a method or slot name as an argument, returns the relevant documentation string. A full description can be obtained using "help all".

Finally, a member function
`int AfterConfigure(Tcl_Interp* interp);`
must be provided. This function will be called after each configure or creation operation, and should return `TCL_OK` or `TCL_ERROR` to indicate whether there is a problem with the configuration.

3 Discussion

Some other recent approaches suggested for integrating Tcl and C++ are more heavy-weight than Objectify, in that they try to provide access to an entire C++ class hierarchy from within Tcl and support automatic argument parsing, or try to handle classes that cannot be modified. I felt that either of these approaches would have been significantly more work than Objectify, and in some cases would have required complete parsing of the classes. When new code is being developed and the programmer has the luxury of defining his classes any way he wants, I feel that the approach described here strikes a good balance between complexity and power.

In addition to simplifying the implementation, using macros to convey information to Objectify makes it possible to support automatic document generation. Documentation is an area of software development that is often given scant and belated attention, in research environments at least, and its maintenance is problematic because the code and the text describing it generally are very loosely coupled. By integrating the code and its documentation the way that Objectify does, this stumbling block is removed. Good class documentation is especially important for Tcl – since it is an attractive choice for a user-level scripting language for many applications, barriers between the programmer and the user can be relaxed somewhat, and if a user is going to write Tcl code to manipulate objects implemented in C++, he must have up-to-date documentation on them.

4 Future work

There are many things that could be done to further ease the integration of C++ objects and Tcl applications. Code could automatically be generated to parse method arguments, possibly using the same sort of mechanism that is used for configuration options. Results could also be automatically formated. The object commands should also allow abbreviations, and some attempt should be made to speed up method dispatch when there are many methods – the use of perfect hashing (e.g., the `gperf` program) has been suggested for this.

The Objectify program is currently written in Tcl, and it is fairly slow, especially for parsing the header files and extracting `TCL_` lines. It should either be written in C, or use an AWK or PERL script for the parsing task.

An interesting extension to Objectify would be to reflect the class hierarchy of a C++ program in the semantics of Tcl objects. Methods and slots could be inherited in a fairly natural way. Currently, Objectify is oblivious to inheritance, but I plan to try and include some support for this in the future.

The Objectify program is available via anonymous ftp from the Tcl archives, in `pub/tcl/code` on the machine `harbor.ecn.purdue.edu` (128.46.128.76). It is freely modifiable and redistributable.

References

[1] John Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1993 (expected).

[2] Donald E. Knuth. Literate programming. *Computer*, 27:97–111, 1984.

[3] The Stone Group. "tclOBST released". Usenet article `1s81fg$d0a@gate.fzi.de`, comp.lang.tcl, May 1993.

[4] Duane Murphy. "Re: Handling C structures". Usenet article `1993May18.171534.27362@novell.com`, comp.lang.tcl, May 1993.