Building Tcl-Tk GUIs for HRT-HOOD Systems

Juan Carlos Díaz Martín
Isidro Irala Veloso
José Manuel Rodríguez García

Abstract

This work explores Tcl-Tk 8.0 as a tool to easily build script based GUIs for Ada95 real-time systems. Tcl-Tk 8.0 is a library that makes graphic programming easier, but it suffers from being not thread-safe. TASH is a thick binding that allows Ada95 single threaded code to use Tcl-Tk. An application architecture is proposed, the deferred server, that provides transparent use of Tcl-Tk to multithreaded Ada95 applications vía TASH. We've found that only a minimal extension to Tcl-Tk 8.0 and TASH is required to support it, and a successful prototype has been implemented based on these ideas. Then, the early integration of Tcl-Tk graphic user interfaces in HRT-HOOD designs is investigated and, unfortunately, we conclude that this is not possible. However, a HRT-HOOD conform distributed configuration is outlined where the user interface becomes a multithreaded remote service based on the deferred server architecture.

1. Introduction

User interface has been a neglectic topic in realtime methodologies in general. However, both in the industrial and the universitary areas, watching and controlling the evolution of the system on development by a friendly graphic environment is important. A good user interface can provide a graphic model of the system evolution, eases its study and understanding and makes its development a more attractive task. This last issue is particularly important in the universitary environment, where time is scarce in order to build a running non trivial system ([Diaz98]) and even more scarce in order to learn and program on a particular windows system. Here, Tcl-Tk scripts appears to be a balanced compromise of power, flexibility and ease of use. In the industry area, besides, in a true hard real time system, the pair (System, User Interface) must be analizable in its temporal constraints.

This article researches: First, a method to use Tcl-Tk scripts from concurrent Ada95 applications and, second, to extend the method to HRT-HOOD systems, by introducing the user interface early in the design stage in order the pair (System, User Interface) to be temporally analizable as a whole. These three main components are shown in the Fig. 1.

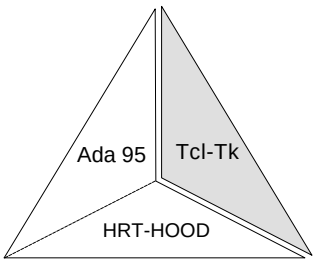

Fig. 1 The main problem components.

As HRT-HOOD automatically translates to Ada 95, the frontier between both is represented by a slim broken line. Unfortunately, Tcl-Tk doesn't directly fit. The Tcl-Tk fitting problem has two sides, namely, the concurrent use of Tcl-Tk and the design restrictictions of HRT-HOOD systems. The rest of the paper is organized as follows. Section 2 summarize Tcl-Tk and TASH and section 3 presents the current work on Tcl-Tk to make it thread-safe and the impact on this paper. The Ada95/Tcl-Tk side is addressed in sections 4 and 5 and the HRT-HOOD/Tcl-Tk

The authors are with the Departamento de Informática, Universidad de Extremadura, Escuela Politécnica, Avda. de la Universidad, s/n, 10071, Cáceres, Spain.

This work was in part supported with U.E. FEDER-II founds (Proyect S742-F96) and Junta de Extremadura.

E-mail: juancarl@unex.es

one in section 6. Section 7 describes our current work for making a distributed GUI for HRT-HOOD systems and section 8 concludes.

2. On Tcl-Tk and Tash

Tcl is a general purpose interpreted command language that admits specialized extensions, being Tk the more known and used. Tk enrich Tcl with a set of commands to use the underlying graphic platform. This augmented Tcl is known as Tcl-Tk. Tk's greatest virtue is probably its ease of use; two or ten lines are enough to get simple applications going. We have written `Adding.tcl`, a Tcl-Tk script that builds the user interface of Fig. 2 takes 26 lines.

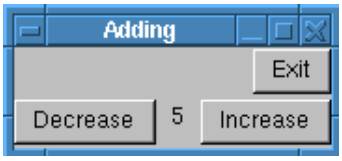

Fig. 2 User interface built by `Adding.tcl`

Tcl-Tk distinguish between two kinds of commands, first, Tcl-Tk *Built-in* commands, such as `button`, and, second, user written commands, such as `increase` and `decrease`, known as *Application Specific Commands*.

As the Tcl-Tk interpreter can be embedded in a C program, our approach is to use the Tcl scripting language in concurrent applications. The Tcl `Tcl_Eval` function allows C code to invoke a Tcl-Tk script. For example, to create the Fig. 2 "Increase" button from a C program, we will use:

```
Tcl_Eval(&Tcl_Interp,
  "button .frame.incr -text Increase \
     -command increase");
```

where `Tcl_Interp` is a handle to a Tcl-Tk interpreter.

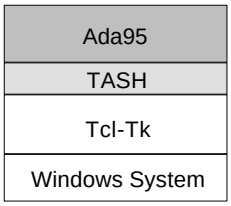

Fig. 3 TASH, a thick binding Ada95/Tcl-Tk

The Tcl-Tk interpreter is written in C and it's easy to call it from a C module by `Tcl_Eval`. To invoke Tcl-Tk from Ada 95, however, is rather cumbersome. TASH (from Tcl-Ada-SHell) is a thick binding Ada95/Tcl-Tk ([West96]). As Fig. 3 shows, TASH provides Ada95 applications with the whole Tcl-Tk interface by the packages `Tcl` and its children.

Callbacks are the Tcl mechanism that implements the interaction of the user with the application. In the Tcl-Tk jargon, a *callback* is a Tcl command `Cmd`, either a user extended command or a Tcl-Tk built-in command, that is passed into another procedure and is executed later, perhaps with additional arguments ([Welc97]). We now introduce the term *user callback*. A user callback is defined as the invocation of an Application Specific Command written in the host language, either C or Ada95, by the Tcl-Tk library. Fig. 4 relates commands, languages, callbacks and user callbacks.

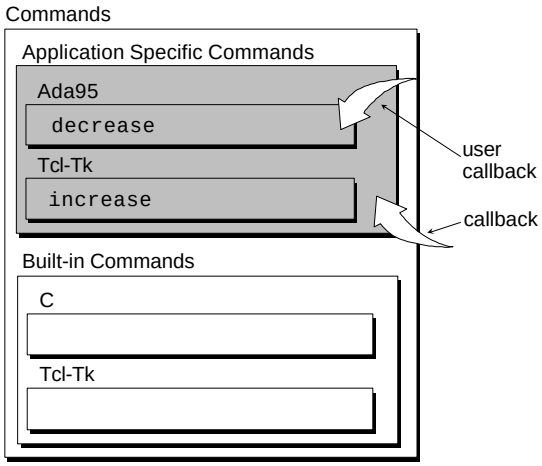

Fig. 4 Languages, commands and callbacks

3. Related work

The initial goal of this work is to build a mechanism to enable any Ada95 task to transparently execute a Tcl-Tk script using `Tcl_Eval`. As of this writing, the Tcl-Tk reentrancy is been addressed by its creators. The upcomming Tcl-Tk 8.1 release have reantrant features. It is thread-safe, though in a rather restricted sense: Each thread can have it's own interpreters but each interpreter can only be accessed by only one thread. Thus, it's true that `Tcl_Eval`, can be concurrently invoked from two different threads, but with (and only with) two separate interpreters. That means that every Ada95 task in the system that

uses a Tcl-Tk GUI should create its own interpreter before any `Tcl_Eval` call, whats complicates programming, but it should not be a mayor difficulty. In order to share a single intepreter, the Tcl-Tk 8.1 provides new Tcl C-level APIs to send commands and scripts to another thread. Of course, the TASH thick binding must keep up with the new thread-safe 8.1 features in a concurrent Ada95 application. Up to that moment, and even later, we think the monolithic approach of this work can be useful. Even it is mandatory for those restricted to use no reentrant Tcl-Tk versions.

4. The deferred server architecture

Service-loop architecture takes the control of the thread that executes it, what prevents Tcl-Tk to be invoked from a multithreaded process. Our effort makes available current Tcl-Tk implementations to concurrent Ada 95 applications in a transparent way.

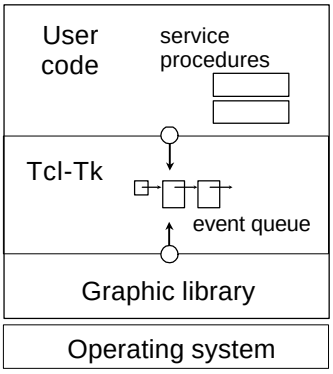

Fig. 5 The Tcl-Tk event queue

In first place, let's analize the Tcl implementation. An ordinary Tcl-Tk program consists of a single thread of control that manages an event queue (Fig. 5), ordered on priority basis. Each event is a data structure comming from two sources, the user code, when `Tcl_Eval` (`Eval` under TASH) is invoked and the graphic library (X-Windows, for example), on user actions. The event structure has two fields, an event identifier and a pointer to a service procedure, either an internal Tcl routine or an user extended command. The thread executes the infinite event-loop `Mainloop`, whose pseudocode is:

```
while(1)
  Do_Event();
```

`Do_Event` access to the Fig. 5 event queue and serves the next event, perhaps by making an user callback. In this case, the invoked Ada95 user extended command may call a Tcl-Tk build-in command, what causes a new event to be inserted in the queue. When the queue gets empty, the thread calls the graphic library, where the whole application gets blocked waiting for user action. Under the X-Windows system, for example, the client library gets blocked on the TCP conection to the server. On return, data are converted into events and inserted in the queue.

In the shown classic Tcl-Tk architecture, the only existing thread takes control, calling the Ada95 Application Specific Commands on user actions. Our purpose is the opposite: Multiple user threads call the Tcl-Tk facilities when needed, i.e., in any time, the `Tcl_Eval` function should be invoked by any thread in order to the interpreter to execute a command. Of course, this goal impose to get rid of the Tcl-Tk infinite service loop. The trouble is that each call to the interpreter has the efect of inserting a new event in the queue and, without the service loop, nobody serves the events. The queue grows endlessly.

Our approach to this issue is to dedicate an aditional *periodic* thread to test and serve the event queue. We pursue not to change the Tcl-Tk event queue model, but rather to chage the way of using it. We have worked on the Tcl-Tk 8.0 version of the library. Two routines have been added to it: `Test_Events` and `Do_Events`. Besides, `Test_Events` and `Do_Events` have been added to the original TASH interface in order to be invoked from Ada95.

`Test_Events` is a fast function that inspects both the event queue and the X-Windows server connection. If there are data in the connection or events in the queue, `Do_Events` is called. `Do_Events` reads all the data in the connection, transforms it in Tcl-Tk events, inserts them in the queue and, finally, serves them. Thus, `Test_Events` and `Do_Events` avoid the blocking service loop provided by the Tcl-Tk. The user code don't use them. Under the new architecture, a new thread tests both the event queue and the server connection on periodic basis. This thread is a dedicated periodic task, `TASH_Handler`, whose implementation is:

```
package TASH_Handler is
  pragma Elaborate_Body;
end TASH_Handler;

with TASH_Controller;
package body TASH_Handler is
  task TASH_Handler_Th is
    pragma Priority(T_H_PRIORITY);
  end TASH_Handler_Th;
  task body TASH_Handler_Th is
    T: Time;
    Period: Time_Span := T_H_PERIOD;
  begin
    T := Clock;
    loop
      if Test_Events then
        Do_Events;
      end if;
      T := T + Period;
      delay until (T);
    end loop;
  end TASH_Handler_Th;
end TASH_Handler;
```

The `delay until` sentence gives the periodic caracter to the task and allows the scheduling of other activities. The period of the task depends on the responsiveness desired from the user interface. The more responsive the application, the smaller the period. Anyway, keeping `Test_Events` fast, the periodic thread interferes a minimum with the rest of the system. The service loop architecture has the advantage of serving the events immediately. Under the proposed design, however, the events wait for `TASH_Handler` to activate. The application tasks use `TASH`, being unaware of `TASH_Handler`. Besides, the Ada95 application tasks see the same interface to the Tcl-Tk script than the classic single-threaded one addressed by TASH. Because of its nature, from now on, we´ll name this design as the *deferred server* (Fig.6).

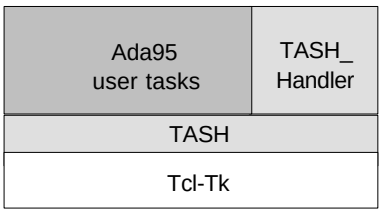

Fig. 6 The deferred server (I)

5. The reentrancy problem

The periodic thread of Fig. 6 solves the service loop problem posed in section 5. The reentrancy problem, however, still remains. Tcl-Tk 8.0 is the typical non thread-safe third party library used by threaded applications. Concurrent invocations of the Tcl-Tk interface cause the **reentrance** in the Tcl-Tk library. The general strategy to yet keep on using Tcl-Tk is to consider the library as a big monolithic monitor.

5.1 The first design attempt

A concurrent application may use the Pthreads interface to support the concurrency. In such a case, the monolithic monitor is implemented as a wrapper on every library access. For example, a multithreaded C program could use Tcl-Tk as follows:

```
pthread_mutex_lock(TclTk_Mutex);
Tcl_Eval(&Interp, "My_Command");
pthread_mutex_unlock(TclTk_Mutex);
```

Fortunately, Ada95 applications using TASH can solve the reentrancy problem much more cleanly (Fig. 7).

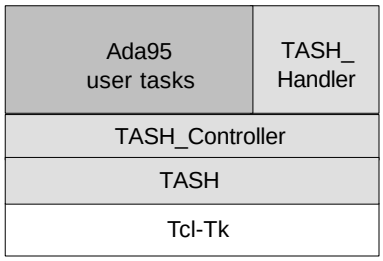

Fig. 7 The deferred server (II)

The key is to introduce a protected object, `TASH_Controller`, that encapsulates the whole TASH interface.

```
with Tcl;
with Tcl.Tk;
package TASH_Controller is
  protected Agent is
    procedure Tcl_Init(...);
    procedure Tcl_CreateCommand(...);
    procedure Tcl_Eval(...);
    procedure Tk_Init(...);
    ...
    procedure Do_Events(...);
    procedure Test_Events(...);
  private
    ...
  end Agent;

  procedure Tcl_Init(...)
    renames Agent.Tcl_Init;
  procedure Tcl_CreateCommand(...);
    renames Agent.Tcl_CreateCommand;
  procedure Tcl_Eval(...);
    renames Agent.Tcl_Eval;
  procedure Tk_Init(...)
    renames Agent.Tk_Init;
  ...
  procedure Do_Events(...);
    renames Agent.Do_Events;
  procedure Test_Events(...);
```

```
    renames Agent.Test_Events;
end TASH_Controller;

package body TASH_Controller is
  protected body Agent is
    procedure Tcl_Init(...) is
    begin
      Tcl.Tcl_Init(...);
    end Tcl_Init;
    ...
  end Agent;
end TASH_Controller;
```

Unfortunately, this attempt of solving the reentrancy problem is faulty. Under a Tcl-Tk interface, the user interacts the system through user callbacks. The rest of this section shows that user callbacks lead to deadlock.

Fig. 8 shows a very simple embedded system, a heater, whose control goal is to keep the water temperature as close as possible to the *r(t)* desired reference.

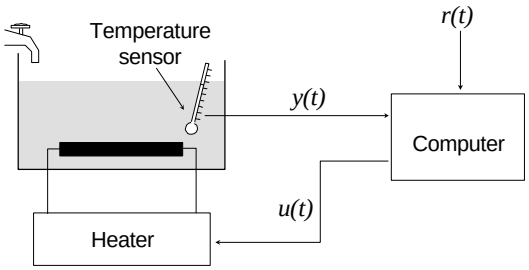

Fig. 8 The heater control system

A Tcl-Tk heater GUI would consist of:

1 A temperature sensor, represented by a Tk widget and updated on periodic basis.
2 An alarm widget that get raised on a critical temperature level.
3 The operator of the system sets the reference temperature *r(t)* by a scale widget.

The set of *user interface primitives* used by an application should be provided in a package object segregated form the rest of the application. From now on, we'll name this object `User_Interface` (Fig. 9).

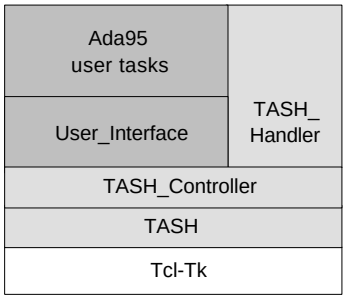

Fig. 9 The deferred server (III)

For example, the heater user interface primitives are `Update_Temp`, used to update the temperature widget, and `Show_Alarm`, used to trigger the alarm widget.

The `User_Interface` package would be:

```
with TASH_Controller;
package User_Interface is
  procedure Initialice_UI(...);
  procedure Update_Temp(...);
  procedure Show_Alarm(...);
end User_Interface;
```

Associated to the scale widget there is an Ada95 user extended command, `Set_Ref_Cmd`, invoked by `Do_Events` on user scale events. `Set_Ref_Cmd`, and in general any user extended command, has two fields of activity as Fig. 10 shows. First, the user code, to update the Ada95 real time variable that holds the reference temperature. The Application Specific Commands should be the only place where operator updates global control variables of the system. Second, the Tcl-Tk library, to show the new reference value on the screen. `Set_Ref_Cmd` uses also `TASH_Controller`, as `User_Interface` entries do.

The question is where to implement the Application Specific Commands, as `Set_Ref_Cmd`. The Ada95 Application Specific Commands aren't ever called from any Ada95 code, but only from the Tcl-Tk library as callbacks. Thus, to put its interface in the specification of `User_ Interface` is harmless, but useless. Our solution is to put them all in a separate package, `User_Commands` as `xx_Cmd` functions (Fig. 10).

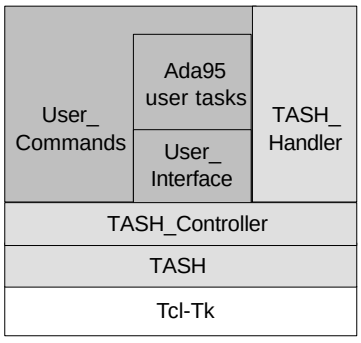

Fig. 10 The deferred server (IV)

Needless to say, the Tcl-Tk library has to know the address of each `xx_Cmd` function in order to invoke it. For example, the Ada95 code

```
cmd := CreateCommands.Tcl_CreateCommand(
  Interp, "decrease",
  Decrease_Cmd'access, 0, NULL);
```

shows how the main procedure pass the address of `Decrease_Cmd` to the Tcl-Tk library. `User_Commands` needs a method, `Create_User_Command`, that let Tcl-Tk know about every `xx_Cmd` function and is invoked by `User_Interface` as part of its initialization. For the heater system, for example, the definition of `User_Commands` would be as follows:

```
with TASH_Controller;
package User_Commands is
  procedure Create_User_Command(
              Interp : Tcl_Interp);
end User_Commands;

package body User_Commands is
 package C renames Interfaces.C;
 package CreateCommands is new
  Tcl.Ada.Generic_Command(Integer);
 function Set_Ref_Cmd (
  ClientData : in Integer;
  Interp     : in Tcl_Interp;
  Argc       : in C.Int;
  Argv       : in CArgv.Chars_Ptr_Ptr
 )return C.Int;
 pragma Convention(C, Set_Ref_Cmd);
 function Set_Ref_Cmd(
  ClientData : in Integer;
  Interp     : in Tcl_Interp;
  Argc       : in C.Int;
  Argv       : in CArgv.Chars_Ptr_Ptr
 )returns C.int is
 begin
   -- 1. Update the Ada95 Reference
   --    Temperature global variable
   -- 2. Show the Reference in the
   --    screen via TASH
 end;
 procedure Create_User_Command(
             Interp : Tcl_Interp) is
 begin
  -- Create Ada95 App. Specific Commands
  -- One entry per User Extended Command
  CreateCommands.Tcl_CreateCommand(
    Interp, "Set_Reference",
    Set_Ref_Cmd'access, 0, NULL);
 end;
end User_Commands;
```

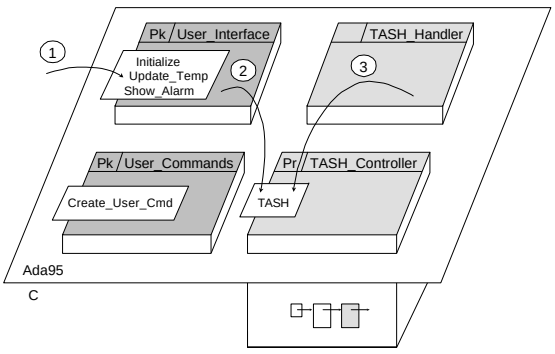

Fig. 11 First design attemp. Succsessful case

Let's see a first working example (Fig. 11):

1. When an Ada95 task decides the temperature widget must be updated, the `Update_Temp` user interface primitive is invoked.
2. `Update_Temp` use the TASH interface encapsulated in the `TASH_Controller` to introduce an event in the queue (the operator doesn't still see any change on the screen).
3. `TASH_Handler` wakes up and invokes `Test_Event`, what causes the invocation of `Do_Event` to serve the event introduced in the step 2, as expected. The new temperature value appears on the screen.

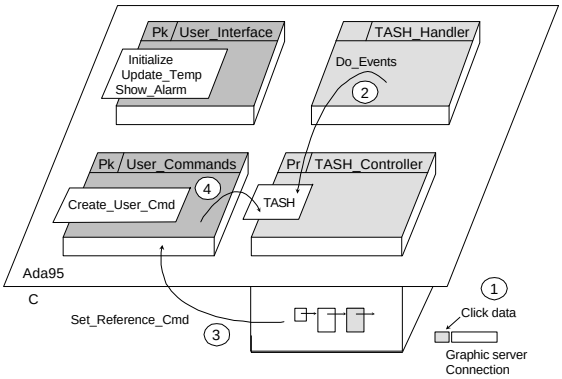

Fig. 12 First design attemp. Faulty case

Fig. 12 illustrates a second example where a user callback takes place:

1. The operator sets the reference temperature through the scale widget via mouse. The graphics library sends the data to the connection (again, the operator doesn't still see any change on the screen).
2. Eventually, `TASH_Handler` wakes up and invokes `Test_Event`, what causes the invocation of `Do_Event` to read the connection, make an event and serve it.
3. To serve the event, `Set_Ref_Cmd` is invoked as an user callback.
4. When `Set_Ref_Cmd` tries to use TASH to put the value on the screen, finds the `TASH_Controller` protected object closed by the current thread. A deadlock has happened. The first try has failed.

5.2 A second design attempt

One solution to the deadlock problem is Application Specific Commands to bypass the protection of `TASH_Controller` by directly invoking TASH. This impose to provide a **double interface** to TASH, the protected one,

TASH_Controller, and the unprotected one, TASH (Fig. 13).

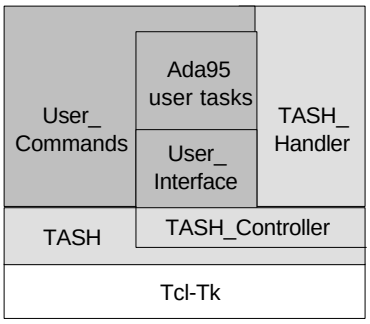

Fig. 13 The deferred server (V)

Under the double interface facility, step 4 doesn´t occasion deadlock because now the thread gets only once through the protection of TASH_Controller. Fig. 14 shows it.

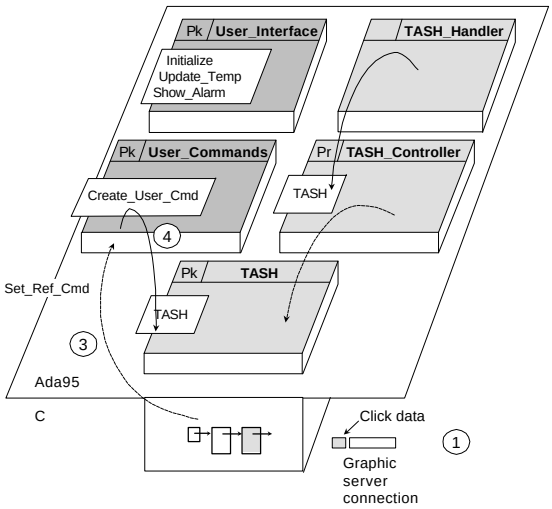

Fig. 14 Second design attemp

Though it seems clear that the double interface is not the ideal solution to the reentrancy problem introduced by user callbacks, it also seems there is not a better one. Notwithstanding, Ada95 applications keep well structured because the solution exhibites a useful property: User_Commands only invokes the raw TASH interface, while User_Interface only invokes the protected TASH_Controller one. Fig. 13 shows the definitive deferred server architecture.

A well known case study, the mine control system ([Burn96]), has been implemented with a Tcl-Tk GUI built upon the deferred server model. We have used Linux 2.0.X, GNAT 3.10p and Tcl-Tk 8.0. Fig. 15 shows a snapshot of the GUI.

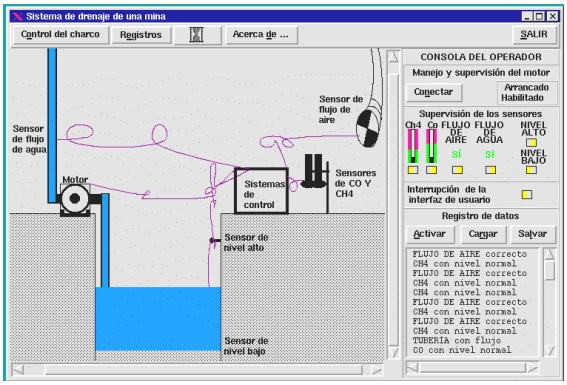

Fig. 15 The Mine Control System Tcl-Tk GUI

6. The HRT-HOOD approach to the deferred server

HRT-HOOD is a well known objet based design metodology that helps in building reliable big realtime systems ([Burn95], [Burn96]). Each HRT-HOOD terminal object has an automatic translation to an Ada95 package with timming attributes. The resulting running system is a concurrent set of Ada95 tasks. Non-terminal objects are called *active*, while there are four kinds of terminal objects: *Passive*, *protected*, *cyclic* and *sporadic*. As an example, the heater system of Fig. 8 is described in HRT-HOOD terms in Fig. 16. Its user interface has been encapsulated in the Operator_Console active object.

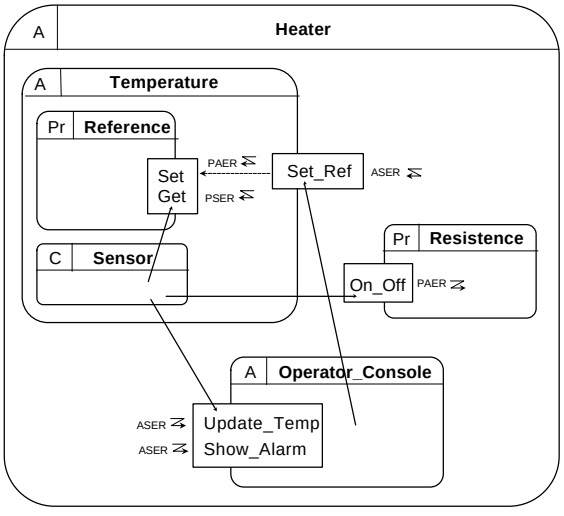

Fig. 16 A first HRT-HOOD design of the heater system

HRT-HOOD requires that the operations on a object delay the invoking thread for a bounded time only. Therefore, the Tcl-Tk written Operator_Console object enforce this rule with operations restricted to be ASER (Asynchronous Execution Request).

Thought HRT-HOOD objects have a direct mapping to Ada95 code, HRT-HOOD specifications ([Burn95]) seems not to be binded to any specific implementation. Therefore, general purpose third-party libraries, such as Tcl-Tk, are not prohibited components in the implementation of any object. What is only relevant is to enforce the HRT-HOOD design restrictions, such as syncronization or timing. The deferred server is a method of encapsulating Tcl-Tk in Ada95 objetcs that can be used by a concurrent application. The question, therefore, is if, without loss of generality, the deferred server architecture can be used to implement the HRT-HOOD `Operator_Console` object of Fig. 16. In the afirmative case, every HRT-HOOD system using a deferred server Tcl-Tk user interface would match Fig. 17.

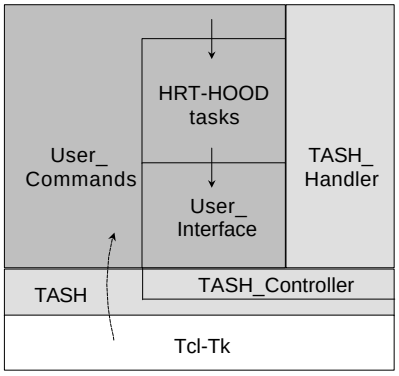

Fig. 17 Hipothetical HRT-HOOD system based on the deferred server architecture

The implementaton of Fig. 17 poses the operating system problem. To achieve predecibility, the HRT-HOOD tasks should live alone, without operating system support. Tcl-Tk, however, need a X-Windows and a Unix box. In order to guarantee predecibility to the real-time tasks, Fig. 18 shows a distributed configuration, where the X-Windows server is moved to other dedicated machine or X-terminal. Now the operating system disappears from the real-time system and the network adapter is controlled by HRT-HOOD objects implemented conforming to the low level programming Ada95 facilities ([Burn96]).

The main shortcoming of this approach is that Tcl-Tk and the X client library must be modified in order to support the operating system services by themselves. This posibility has been explored. The single line Tcl script

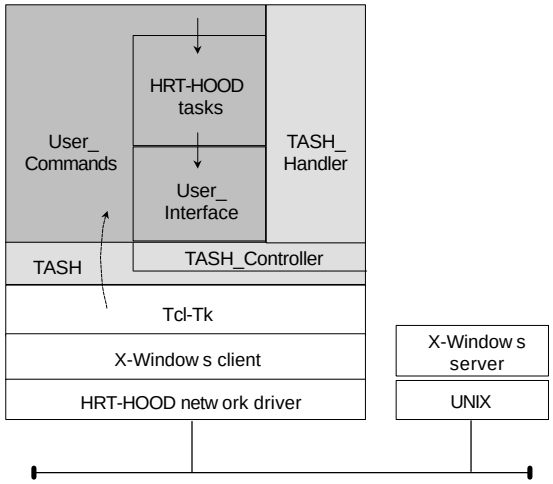

Fig. 18 Distributed HRT-HOOD/Tcl-Tk system (I)

```
#!/usr/bin/tclsh
puts stdout "Hello, world"
```

makes 18 different system calls and a total number of 86. The 26 lines Tcl-Tk script `Adding.tcl` that builds Fig. 2 GUI makes 25 different system calls and a total number of 488 ones. It's true that the system is now fully analizable in its temporal requirements, but Tcl-Tk and the client component of X-Windows entrust functionality to the operating system that is difficult to assume by themselves and the Ada95 run-time system.

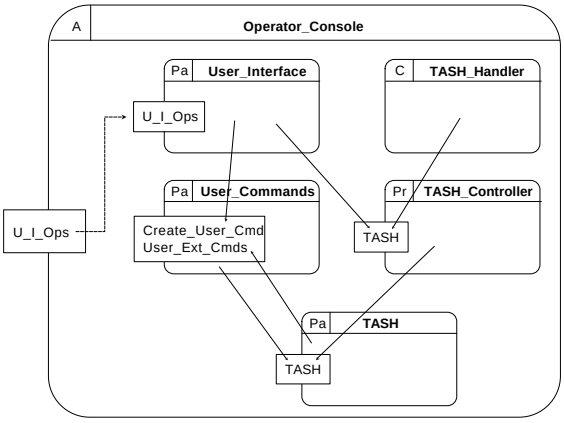

Fig. 19 A faulty HRT-HOOD design of the Tcl-Tk User Interface

The operating system problem, however, is not the only one. The Fig. 19 is a refinement of `Operator_Console` into terminal objects. As can be infered by the objects names and relations, the goal of this decomposition is to map automatically into the Ada95 deferred server architecture proposed in previous sections. Some design and implementation problems can be identified on it:

First, HRT-HOOD terminal objets must be temporally analizable. A HRT-HOOD protected object has two attributes, the ceiling priority and the worst case execution time (WCET). The last one is the worst case execution time of its slower operation and determines the bloking imposed to the invoking tasks. As the HRT-HOOD protected object `TASH_Controller` is implemented as the Tcl-Tk library, to wich it guarantees mutual exclusion, its very difficult to determine its WCET parameter, due to the unpredictable operating system time response.

Second, HRT-HOOD specification ([Burn95], pg. 29) explitely "forbids passive (or protected) objects to use each other in a cyclic manner". Tcl-Tk user callbacks necessarily introduce a cycle between `User_Commands` and `TASH`.

Third, HRT-HOOD specification ([Burn95], pg. 30) explicitely sets that passive objects can only invoke operations on passive objects. Passive `User_Interface` invokes protected `TASH_Controller`.

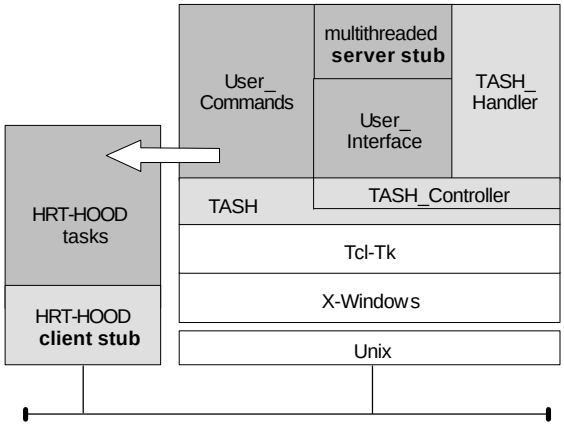

Fig. 20 Distributed HRT-HOOD/Tcl-Tk system (II)

The above run-time analisys and the inherent violations of HRT-HOOD rules described above make clear that to embed Tcl-Tk in a HRT-HOOD system is not possible. Therefore, other alternatives must be explored. Fig. 20 shows a distributed configuration where the deferred server goes to a second machine, as a remote service to the HRT-HOOD system. The interface definition of this service consists of the user interface primitives of the real-time system. As Fig. 20 shows, the HRT-HOOD tasks are substituted in the server side by a multithreaded server stub.

User callbacks are problematic. What does happen when a user extended command accesses a real-time variable? Now the HRT-HOOD system is not only a *client* of Tcl-Tk services, but it becomes a *server* of its real-time variables, monitorized and controlled by the operator. Notwithstanding, the `Operator_Console` object gets much lighter that the one of Fig. 19. It implements the client stub by taking the control of the network adapter. Its operationes are ASER in order to bound the delay imposed to real-time tasks: They return when the packet has been put in the network. It also implements the real-time variables service stub by introducing a thread that listen in the network adapter for real-time requests. As an example, Fig. 21 shows the operator console object for the heater system.

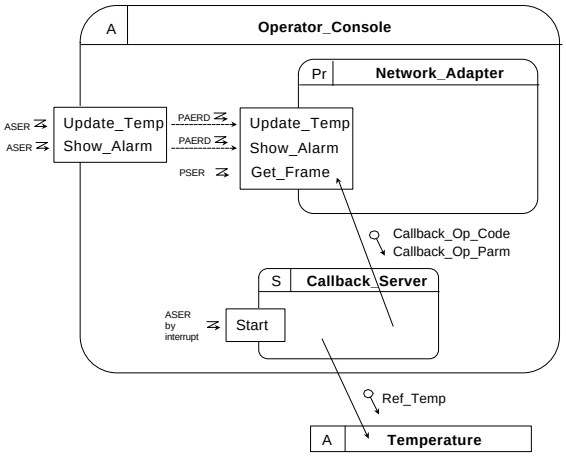

Fig. 21 A second HRT-HOOD design of the Tcl-Tk User Interface

7. Work under way

Whether the deferred server arquitecture has been implemented and tested by itself in a centraliced way, as a remote service of a HRT-HOOD system, not yet. The main problem to be solved is the communication protocol. TCP/IP is a good ellection in the Tcl-Tk side, but may be considered big and heavy in a HRT-HOOD side. Three possibilities are been explored. First, buiding an analizable TCP/IP server active object in the HRT-HOOD system based on the Minix user space implemented TCP/IP server. Second, to unload the HRT-HOOD side from any communication protocol by faking the Tcl-Tk side inserting/discarding the frames TCP/IP stuff. Third, use a much ligther network protocol, FLIP ([Tane95]).

8. Conclusions

First, the problem of using the Tcl/Tk scripting language in mutithreaded Ada95 applications has been studied and a solution has been proposed: the deferred server architecture, that demands a little extension to Tcl-Tk/TASH. A complex enough classic example, the mine control system, has been implemented to test these ideas with success. Second, the integration of Tcl/Tk in HRT-HOOD systems has been explored. The reentrant nature of Tcl-Tk and its mechanism of callbacks takes us to conclude that is not possible to buid a stand alone true hard HRT-HOOD system with a Tcl-Tk user interface. Finally, a distributed approach to the Tcl-Tk GUI that relies on the deferred server model, however, seems promising. This distributed architecture has been presented and its problems posed. Work is in progress in order to gain experience on it and to achieve the goal of developing true HRT-HOOD systems with Tcl-Tk script based user interfaces.

9. References

[Barn95] Barnes, J., *Programing in Ada95*, Addison-Wesley, 1995.

[Burn95] Burns, A. and Wellings, A., *HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems*, Elsevier, 1995.

[Burn96] Burns, A. and Wellings, A., *Real-Time Systems and Programing Languages*, Addison-Wesley, 1996.

[Diaz98] Díaz Martín, J.C., Irala Veloso, I., "Prácticas de Sistemas de Tiempo Real en la Uex. Integración de Tcl-Tk en HRT-HOOD", *Jenui'98, Actas del congreso, pp. 166-172, Escola d'Informática d'Andorra. Sant Juliá de Lória, Andorra*, July, 9-10, 1998

[IEE96] IEEE, "Information Technology - Portable Operating System Interface (POSIX)- Part 1: System Application Program Interface (API) [C Language], IEEE Std 1003.1, 1996 Edition, (1996).

[Oust94] Ousterhout, John, *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, 1994.

[Tane95] Tanenbaum, A. S., *Distributed Operating Systems*, Prentice-Hall, 1995.

[Welc97] Welch, B., *Practical Programming in Tcl & Tk*, Prentice-Hall, 1997.

[West96] Westley, T., “TASH: Tcl Ada Shell, An Ada/Tcl Binding”, *ACM SIG Ada Ada Letters*, 1996.