Prácticas de Sistemas Tiempo de Real en la Ingeniería Informática de la Uex: Integración de TCL-TK en HRT-HOOD

Juan Carlos Díaz Martín
Isidro Irala Veloso

Escuela Politécnica. Avd. de la Universidad, S/N
10071 Caceres
juancarl@unex.es

***Resumen**: Este artículo versa sobre la asignatura Sistemas de Tiempo Real en el plan de estudios de Ingeniería Informática en la Universidad de Extremadura. Presenta el programa básico de la asignatura y abunda en la parte práctica: La construcción de un sistema de tiempo real utilizando la metodología HRT-HOOD. HTR-HOOD es discutida desde el punto de vista de la praxis académica: la ausencia de primitivas gráficas en la definición del lenguaje Ada95 dificulta la implementación de la interface de usuario, hace a la práctica laboriosa en exceso y la resta actractivo para el estudiante. El artículo aborda el problema mediante TCL-TK. Primero, propone cómo introducir la interface gráfica de usuario ya en la etapa de diseño HRT-HOOD como uno de sus cinco objetos básicos. Segundo, explica cómo en la etapa de programación se construye la interface de usuario enteramente en Ada95. Para ello hacemos uso de una nueva herramienta gráfica multiplataforma de libre distribución que integra Tcl-Tk y Ada95: TASH.*

1. Programa de la asignatura

En la Escuela Politécnica de Cáceres se imparten tres titulaciones de Informática: Ingeniería Informática (II), Ingeniería Técnica en Informática de Sistemas (ITIS) e Ingeniería Técnica en Informática de Gestión (ITIG). Fueron publicadas en BOE en Enero de 1.994.

Sistemas de Tiempo Real es una asignatura optativa de la II. Se imparte en el segundo cuatrimestre del quinto curso. Tiene un peso de 4.5 créditos teóricos y 1,5 prácticos.

El programa teórico de Sistemas de Tiempo Real se apoya fundamentalmente en la segunda edición del libro de texto de Burns y Wellings ([4]) "Real Time Systems and Programming Languages". Como ya lo fue en la primera edición del mismo, esta libro está llamado a ser la referencia principal en textos universitarios en la materia.

El libro aborda todos los tópicos de la disciplina utilizando varios lenguajes de tiempo Real: Ada95, Modula, Occam2, CHILL, Mesa, C++ y C-POSIX. El programa de la asignatura utiliza sólo dos, Ada95 y C-POSIX. Las razones son las siguientes:

- En nuestra opinión, tantos lenguajes son demasiados para una asignatura cuatrimestral.
- Queremos dar a la asignatura un carácter práctico utilizando herramientas software de libre distribución. Afortunadamente, para Ada95 y C-POSIX, estas herramientas sí están disponibles.
- Ada95 ([5]) es un lenguaje nuevo que recoge la experiencia de los últimos años aquirida en la práctica y en la investigación en ingeniería de sistemas empotrados grandes. Es un lenguaje muy grande, con suficientes contenidos de la disciplina de tiempo real como dedicarle, por sí solo, la asignatura completa.
- No puede ignorarse en una asignatura universitaria sobre Tiempo Real las dos facilidades del estándar POSIX: Pthreads y la extensión de tiempo real.

Con estas premisas, el programa teórico de la asignatura (4,5 créditos) se conforma como sigue:

1. Introducción
2. Programación de sistemas grandes
3. Tolerancia a fallos

4. Excepciones
5. Programación concurrente
6. Tiempo Real
7. Planificación
8. Manejadores de Dispositivo
9. La metodología HRT-HOOD
10. POSIX y Tiempo Real

Se dispone de 1,5 créditos para implementar las prácticas. Se ha optado por trabajar sobre la metodología HRT-HOOD. Las razones son:

- Se dispone de las herramientas software adecuadas, fundamentalmente del compilador de Ada95 de libre distribución GNAT 3.10p.
- Proporcina al alumno la oportunidad de desarrollar un sistema real completo en el que identifica sus requisitos temporales y en el que trabaja utilizando un verdadero lenguaje de tiempo real.
- El ser una extensión de HOOD da un carácter formal y metodológico al ciclo de desarrollo.
- HRT-HOOD proporciona una traducción sistemática de objetos básicos HRT-HOOD a código Ada95.

2. El entorno de la asignatura

La titulación de II en la Uex es nueva. La asignatura se ha impartido durante dos cursos académicos: 96/97 y el presente, 97/98. Por lo tanto la experiencia es escasa.

La matriculación es baja, 15 alumnos en el curso 97/98, de modo que el contacto con el alumno es estrecho.

Se dispone del material de prácticas adecuado. Las prácticas se imparten en una sala de 30 PC's con Windows95 y Linux Red-Hat 5.0.

Se dispone del compilador GNAT 3.10p para el primero. La distribución Windows95 incorpora un entorno de desarrollo de Ada95 de alta calidad: AdaGuide.

3. La Metodología HRT-HOOD

A nuestro juicio, son dos las novedades más interesantes de la segunda edición del libro de Burns and Wellings ([4]): Los capítulos sobre planificación de Tiempo Real y el caso práctico resuelto mediante la metodología HRT-HOOD. HRT-HOOD significa diseño jerárquico orientado a objetos de sistemas críticos de tiempo real (*Hard Real Time Hierarchical Object Oriented Design*). Se centra en el diseño de la arquitectura lógica y física del sistema y usa una notación basada en objetos.

En la asignatura utilizamos una versión simplificada de HRT-HOOD y la aplicamos a un caso de estudio muy utilizado en la literatura de Tiempo Real: El sistema de control de una mina:

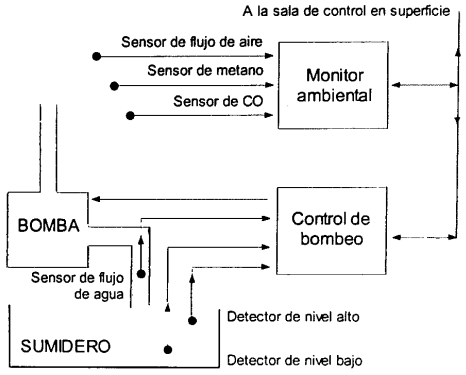

Fig. 1 Caso práctico HRT-HOOD: La mina

La Fig. 2 muestra el sistema de la mina descrito en términos de cuatro objetos no terminales HRT-HOOD en un primer nivel de descomposición.

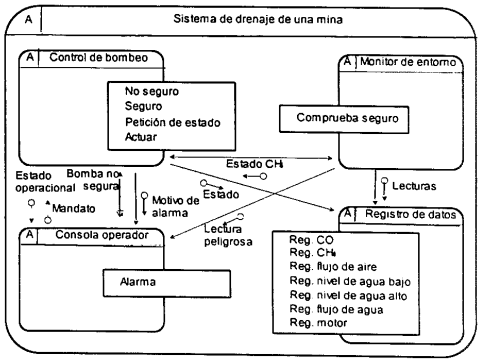

Fig. 2 Primer nivel de descomposición HRT-HOOD

HRT-HOOD utiliza cinco tipos de objetos. Los objetos no terminales no son analizables en cuanto a sus requisitos temporales y se denominan activos. Los objetos terminales sí son analizables y tienen una traducción sistemática a Ada95. Hay cuatro tipos: pasivos, protegidos, periódicos y esporádicos. La Fig. 3 muestra uno de los objetos hijos no

terminales del objetos Control de Bombeo de la Fig. 2: Sensor Nivel Agua. Este se descompone a su vez en dos objetos terminales. Uno protegido y otro esporádico.

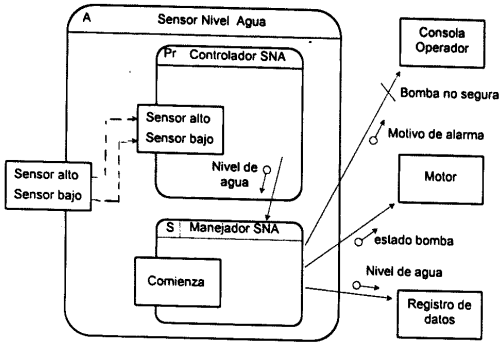

Fig. 3 Objeto activo con dos objetos terminales hijos

El objeto Manejador SNA arranca la bomba y apaga la bomba en respuesta a la interrupción de nivel de agua (bajo o alto). Cada objeto terminal se implementa como un paquete del mismo nombre. Los atributos de tiempo real de este objeto y su interface se generan automáticamente a partir de los requisitos no funcionales y el diseño HRT-HOOD. Además, el cuerpo tiene una pauta bien establecida. Por ejemplo:

- Atributos de tiempo real

```
with System; use System;
package Manejador_SNA_Rtatt is
  Ceiling_Prio : constant := 11;
  Thread_Prio : constant Priority:=6;
end Manejador_SNA_Rtatt;
```

- Interface del objeto

```
with Manejador_SNA_Rtatt;
use Manejador_SNA_Rtatt;
package Manejador_SNA is -- ESPORÁD
  type Nivel is (High, Low);
  procedure Comienza(Int : Nivel);
end Manejador_SNA;
```

4. El alumno frente a HRT-HOOD: problemas planteados

En la clase de prácticas, al alumno se le propone un caso que debe implementar en HRT-HOOD. Este caso, como el de la mina explicado en teoría, es un sistema que:

- Es complejo, ya que tiene varios threads.
- Evoluciona según el entorno que controla
- Evoluciona con el tiempo de forma continua

Desde un punto de vista didáctico se hace evidente la necesidad de que el alumno disponga de un **modelo gráfico** del sistema construido para seguir la evolución de componentes a lo largo el tiempo.

Ada95 sólo dispone de paquetes de texto con funcionalidad básica. Ni siquiera existe una función con coordenadas de pantalla. En nuestra opinión, la carencia de una interface gráfica fácil de usar reduce el interés de dedicar los créditos prácticos a trabajar sobre HRT-HOOD.

Reconociendo el interés de HRT-HOOD, la solución pasa por disponer de una biblioteca gráfica para Ada95 de libre distribución. Una de ellas es AdaGraph, escrita para Windows95 y poco potente ya que dispone de primitivas demasiado básicas (círculos, polígonos, líneas), lo que conlleva un desarrollo lento y laborioso del modelo. En Linux podemos importar el Api de X-Window u otro API sobre X-Lib. El problema es que estos API son de nuevo excesivamente complejos para el tiempo disponible en las prácticas. Resumiendo, la solución escogida es dependiente de la plataforma y difícil de utilizar.

5. Una herramienta gráfica multiplataforma: TCL-TK

Tcl-Tk es un lenguaje de mandatos con las siguientes características ([1, 2]):

Gratuito: El código fuente es propiedad de la Universidad de California y Sun Microsystems y está libremente disponible en Internet

Interpretado: Un programa Tcl-Tk se interpreta, aunque versiones compiladas están en proyecto en Sun Microsystems

De propósito general: Proporciona facilidades para la programación como variables, procedimientos y estructuras de control

Extensible: Tcl puede extenderse con nuevos mandatos escritos en Tcl y en C, incluso en tiempo de ejecución. La extensión más conocida es Tk, que proporciona una interface a X-Window

Empotrable: Su intérprete es una biblioteca de funciones C que puede ser incorporada en cualquier aplicación y extendido por esta.

Portable: A partir de Tcl7.5 y Tk4.1, Tcl-Tk está disponible para Unix, Mac, Windows95...

Algunas ventajas de usar Tcl-Tk en una aplicación son:

Desarrollo rápido: Tcl-Tk está disponible como un lenguaje de más alto nivel que C o Ada95 y, como es interpretado, acelera el ciclo edita-compila-enlaza-prueba del desarrollo del software.

Integración de sistemas: Es un lenguaje excelente para integrar varios programas en una única interface de usuario. Como es empotrable, puede actuar como un lenguaje interaplicación.

Programación de usuario. El usuario puede programar directamente en Tcl-Tk. Muchas aplicaciones de éxito como AutoCad y Excel están diseñadas como un lenguaje de mandatos. La ventaja de usar Tcl-Tk es que no es necesario idear un nuevo lenguaje.

Las desventajas de Tcl-Tk son que no existe el tipado de datos, la modularidad es muy limitada y es pobre la gestión del espacio de nombres. Estos problemas son irelevantes para nosotros, ya que no pretendemos desarrollar código en Tcl-Tk, sino aprovechar sus facilidades gráficas.

6. TASH: Integrando Ada95 y Tcl-Tk

Tcl-Tk se escribió en lenguaje C para ser utilizado junto con el lenguaje C. TASH ([2]) es una nueva herramienta que enlaza Ada 95 con Tcl-Tk. Permite usar Ada95 en lugar de C para:

1. Implementar mandatos Tcl-Tk en Ada95
2. Invocar mandatos Tcl-Tk desde código Ada95.

Ada95 facilita el importar código escrito en C. Básicamente, TASH es un conjunto de ficheros especificación Ada95 que declara la interface de Tcl-Tk. El código fuente de TASH se distribuye para ser recompilado en cualquier sistema que soporte Tcl-Tk y GNAT. Ello significa que una aplicación TASH ejecuta en Linux o Windows95 sólo vía recompilación.

La arquitectura de un programa Ada95 que empotra Tcl-Tk es el siguiente (Fig. 4):

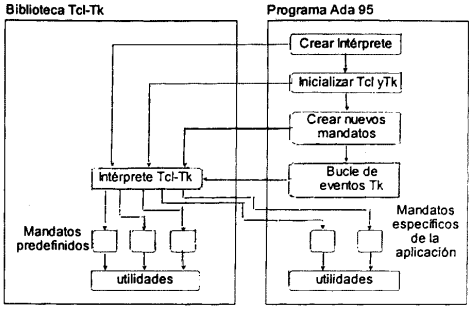

Fig. 4: Arquitectura de un programa Ada95 con Tcl-Tk

La Fig. 5 presenta un programa sencillo escrito en lenguaje Tcl-Tk, `"./entorno.Tcl"`, que construye la siguiente interface de usario:

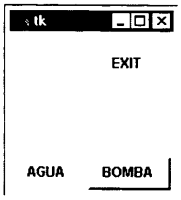

Fig. 5: La interface de usuario

```
frame .menu -borderwidth 10
button .menu.exit -text EXIT -command exit
pack .menu.exit -side right
pack .menu -side top -fill x
frame .separador -width 100 -height 50
pack .separador
frame .body -bg grey50
pack .body -side bottom
button .body.bomba -text BOMBA -command bomba
pack .body.bomba -side right
button .body.agua -text AGUA -command agua
pack .body.agua -side  left
```

Al pulsar AGUA o BOMBA el intérprete Tcl devuelve error porque los mandatos específicos de la aplicación (Fig. 4) asociados a los botones no están dados de alta en el intérprete original.

Ahora vamos a empotrar el guión Tcl-Tk anterior en un programa Ada95, `EjemploAda95` y el código de los botones vamos a escribirlo como un procedimiento convencional del programa. En aras de la simplicidad, en este ejemplo ambos procedimientos, `Agua_Command` y `Bomba_Command` coinciden:

```
with CArgv;
with CHelper;
with Interfaces.C.Strings;
with Tcl;      use Tcl;
with Tcl.Ada; use Tcl.Ada;
with Tcl.Tk;
with Text_IO;

procedure EjemploAda95 is
  package C renames Interfaces.C;
  package CreateCommands is
          new Generic_Command (Integer);
  function "=" (Left, Right : in C.Int)
     return Boolean renames C."=";
  Interp       : Tcl_Interp;
  Command      : Tcl_Command;

  function Agua_Command (
    ClientData : in Integer;
    Interp     : in Tcl_Interp;
    Argc       : in C.Int;
    Argv       : in CArgv.Chars_Ptr_Ptr
    return C.Int;

  pragma Convention (C, Agua_Command);
  -- Declaración de mandato Tcl

  function Agua_Command (
    ClientData : in Integer;
    Interp     : in Tcl_Interp;
    Argc       : in C.Int;
    Argv       : in CArgv.Chars_Ptr_Ptr
  ) return C.Int is
  begin
    Tcl_EvalFile(interp,"./comando.Tcl");
    return TCL_OK;
  end Agua_Command;

  function Bomba_Command (
    ClientData : in Integer;
    Interp     : in Tcl_Interp;
    Argc       : in C.Int;
    Argv       : in CArgv.Chars_Ptr_Ptr
  ) return C.Int;

  pragma Convention (C, Bomba_Command);
  -- Declaración de mandato Tcl

  function Bomba_Command (
    ClientData : in Integer;
    Interp     : in Tcl_Interp;
    Argc       : in C.Int;
    Argv       : in CArgv.Chars_Ptr_Ptr
  ) return C.Int is
  begin
    Tcl_EvalFile(interp,"./comando.Tcl");
    return TCL_OK;
  end Bomba_Command;

begin -- Comienza el programa

  -- Creación de un intérprete Tcl
  Interp := Tcl_CreateInterp;
  -- Inicialización de Tcl
  if Tcl_Init (Interp) = TCL_ERROR then
    Text_IO.Put_Line ("Mina: Tcl_Init failed: "
     & Tcl.Ada.Tcl_GetResult (Interp));
    return;
  end if;

  -- Inicialización de Tk
  if Tcl.Tk.Init (Interp) = TCL_ERROR then
    Text_IO.Put_Line ("EjemploAda95:
                       Tcl.Tk.Init falló: "
      & Tcl.Ada.Tcl_GetResult (Interp));
      return;
  end if;

  -- Dar de alta en el intérprete los mandatos
  -- anteriores escritos en Ada95
  Command := CreateCommands.Tcl_CreateCommand (
      Interp, "agua",  Agua_Command'access, 0,
      NULL);
  Command := CreateCommands.Tcl_CreateCommand (
      Interp, "bomba",  Bomba_Command'access,
      0, NULL);

  --Código Tcl-Tk: La interface de usuario
  Tcl_EvalFile(interp,"./entorno1.Tcl");

  -- Bucle infinito de servicio.
  Tcl.Tk.MainLoop;

end EjemploAda95;
```

Como se aprecia, los procedimientos Agua_Command y Bomba_Command se limitan a invocar el intérprete Tcl para que procese el fichero fuente Tcl-Tk "./comando.Tcl":

```
tk_messageBox -type ok -message
        "EJECUCION DE UN COMANDO ADA"
```

El resultado es el cuadro de diálogo:

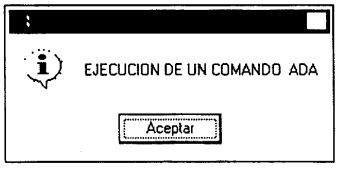

Fig. 6: Cuadro de diálogo

EjemploAda95 muestra la sencillez con la que se construye una interface de usuario en Ada95 haciendo uso de TASH. El único precio a pagar es por el estudiante es conocer algunos mandatos Tcl-Tk.

7. Integración de TASH en HRT-HOOD: problemas planteados

Dado el estado actual de desarrollo del intérprete Tcl-Tk, al menos en su versión pública actual, la 8.0, es una aplicación de un solo thread de control. Por lo tanto, el programa anterior en Ada95 EjemploAda95 es también, desgraciadamente, una aplicación de un solo thread de control. El procesador está dedicado en la sentencia Ada95

`Tcl.Tk.MainLoop` a la espera y el servicio de eventos producidos por la interface de usuario y por la aplicación según muestra el código interno de Tk:

```
void Tk_MainLoop()
{
  while (Tk_GetNumMainWindows() > 0) {
    Tcl_DoOneEvent(0);
  }
}
```

y el código de TASH

```
with CArgv;
package Tcl.Tk is
   ...
   procedure MainLoop;
   pragma Import (C, MainLoop, "Tk_MainLoop");
   ...
private
   ...
end Tcl.Tk;
```

En cada iteración del bucle `Tcl.Tk.MainLoop`, `Tcl_DoOneEvent`:

1. Realiza un escrutinio de los eventos pendientes en el sistema operativo (pulsación de teclas, movimientos de ratón) y los sirve. Por ejemplo, ejecutando el código asociado a un botón recién pulsado.
2. Examina si existe alguna petición de mandato Tcl-Tk de la aplicación y le da curso.
3. Si no hay eventos pendientes, suspende la aplicación

El control de la aplicación por parte de `Tcl.Tk.MainLoop` imposibilita su uso en una aplicación multithread Ada95. Una aplicación HRT-HOOD es multithread. Cada objeto activo HRT-HOOD contiene al menos un thread de control. En conclusión, dado su estado actual de desarrollo, Tcl-Tk no se puede utilizar en las prácticas de Sistemas de Tiempo Real para realizar proyectos HRT-HOOD.

Esta es la situación que ha motivado este trabajo. En lo que sigue, presentamos una solución parcial al problema.

8. Una primera solución: El servidor diferido

La introducción de una interface de usuario Tcl-Tk en un sistema HRT-HOOD exige un nuevo objeto activo que vamos a denominar "Interface de Usuario". La función del objeto "Registro de Datos" de la Fig. 2 es simplemente registrar el evento en una base de datos. La función de "Interface de Usuario" es simétrica: recoge cada evento producido en el sistema para invocar el mandato Tcl-Tk que modifica actualizar la interface gráfico de usuario. El objeto es el siguiente:

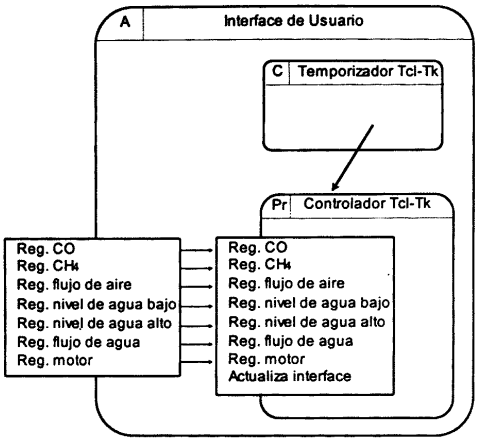

Fig. 7: El nuevo objeto "Interface de Usuario"

El objeto "Controlador Tcl-Tk" es un objeto protegido que se ocupa de invocar los mandatos correspondientes a cada uno de sus métodos. Hay que tener en cuenta que la ejecución de un mandato Tcl-Tk no actualiza inmediatamente lo que el usuario u operador del sistema observa en el monitor. Para que se procuzca este efecto es preciso invocar la función `Tcl_DoOneEvent`. De ello se encarga el método "Actualiza Inerface" de la figura 7. Este método es invocado por el objeto cíclico "Temporizador Tcl-Tk" de forma periódica. A este diseño se le denomina en la literatura de tiempo real "Servidor diferido".

La ventaja de este esquema es que es fácil de implementar y proporciona la interacividad del operador con la interface de usuario. Por ejemplo, el operador puede redimensionar la interface de usuario mientras el resto de tareas del sistema sigue ejecutando en concurrencia. La desventaja es que el tiempo de respuesta del sistema hacia el operador es dependiente del periodo otorgado al objeto "Temporizador Tcl-Tk". El periodo puede reducirse para mejorar la interactividad, pero en esa misma medida aumenta la carga del sistema y los plazos de las tareas críticas se ven afectados.

La interface de usuario de la aplicación (Fig. 5) se construye en el bloque de inicialización del objeto "Temporizador Tcl-Tk". Su evolución temporal se realiza en el objeto "Controlador Tcl-Tk".

9. Conclusiones

El uso de TASH en la construcción de interfaces gráficos de usuario de sistemas de tiempo real ha sido discutido. Se ha propuesto un método para introducir TASH en la metodología de diseño de sistemas de tiempo real HRT-HOOD respetando sus objetos básicos.

Es preciso disponer cuanto antes de una versión multithread de la biblioteca Tcl-Tk que no exija a la aplicación que la usa la atención a los eventos de la operador, con ello se eliminarían los problemas asociados al servidor diferido.

Uno problema que queda abierto es el análisis de tiempo de respuesta de la biblioteca Tcl-Tk. de los tiempos de servicio de los mandatos Tcl-Tk. Por ejemplo, si un mandato se implementa como un guión interpretado almacenado en disco, los tiempos de respuesta se disparan. Una versión Tcl-Tk de tiempo real sería también bienvenida.

10. Referencias

1. Ousterhout, John. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, MA, 1994.
2. Westley, Terry. "TASH: Tcl Ada SHell, An Ada/Tcl Binding." *ACM SIGAda Ada Letters*, 1996.
3. Welch, B., *Practical Programminig in Tcl-Tk*, Prentice-Hall, 1997
4. Burns, A. and Wellings, A., *Real-Time Systems and Programming Languages*, Addison-Wesley, 1996
5. Barnes. J., *Programming in Ada95*, Addison-Wesley, 1995