Documentación

Proyecto Rhino

Marco Alvarez Reyna
CIII
UTN-FRC

2008

Índice de contenido

Documentación del Proyecto Rhino

1 Entorno: Desarrollo

1.1 Plataforma:

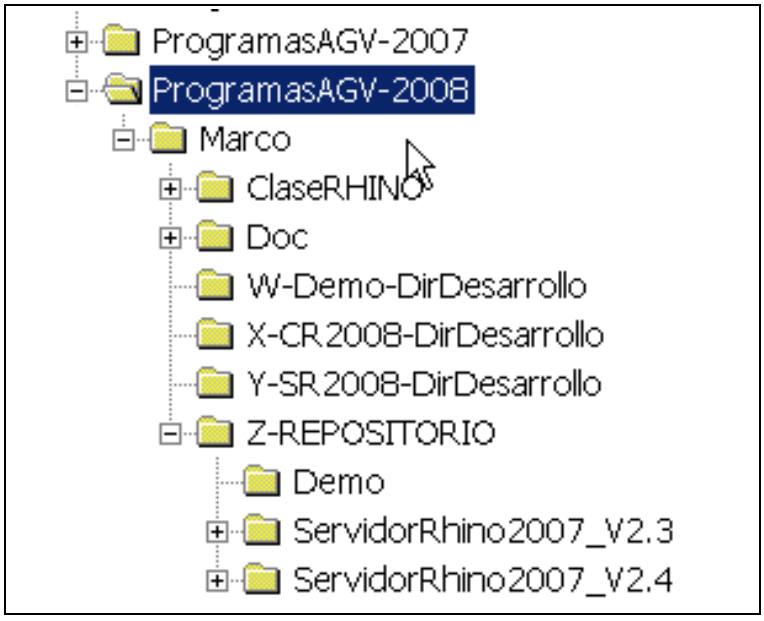
El Cliente y el Servidor del sistema *Servo Robot Rhino XR4* fueron desarrollados en *Borland Builder C++ 5.0/6.0,* utilizando *Windows 2000* como sistema operativo. La plataforma de hardware utilizada para el desarrollo de software, como así también para ensayos con el servo robot se baso en tecnología *X86 32bit*.

La clase Rhino fue desarrollada bajo el entorno *SUSE Linux 9.0/10.0*, utilizando e*macs* como editor y *gcc (g++)* como compilador. La misma es 100% portable Linux/Windows. Puede incorporarse a un proyecto en *Borland Builder C++ 5.0/6.0,* compilar y correr sin ningún cambio previo (probado)
Como herramienta de debug se utilizo *DDD*, y *SourceNav* para inspeccionar el código. Para la documentación se utilizo *doxygen*.

Para que cualquiera de las aplicaciones pueda ser ejecutada, es necesario contar con una instalación de *Borland Builder C++ 6.0* con los objetos correspondientes (Indy, TcomPort, Joystick)

1.2 Estructura de directorios:

Los directorios de trabajo se organizaron según la estructura mostrada a continuación. Todos los archivos de cada proyecto se encuentran bajo el mismo directorio, a excepción de los archivos de la clase Rhino que se encuentran en un directorio independiente.



1.2.1 Archivos relacionados:

1.2.1.1 Dir-Info.txt (/ProgramasAGV-2008/Marco/Dir-Info.txt)

Este archivo contiene información sobre los directorios de trabajo

```
PROYECTO RHINO

Organización de los directorios de trabajo:

**************************ANO 2008*********************************

Directorios bajo "C:\ProgramasAGV-2008\Marco":

- El directorio "W-Demo-DirDeDesarrollo" es el directorio de desarrollo del programa de
demostración.

- El directorio "X-CR2008-DirDeDesarrollo" es el directorio de desarrollo del cliente.

- El directorio "Y-SR2008-DirDeDesarrollo" es el directorio de desarrollo del servidor.

- El directorio "Z-REPOSITORIO" contiene las versiones estables de los programas en
desarrollo.

- El directorio "ClaseRHINO" contiene el proyecto: clases Rhino, portable Linux/Windows
(SuSE 9.0/10.0)

- El directorio "Doc" contiene la documentación del proyecto.

*****************************************************************

Marco Alvarez Reyna
CIII UTN-FRC Argentina
marcoalrey@gmail.com
```

2 **Desarrollo Cliente/Servidor**: Control

2.1 Cliente Rhino

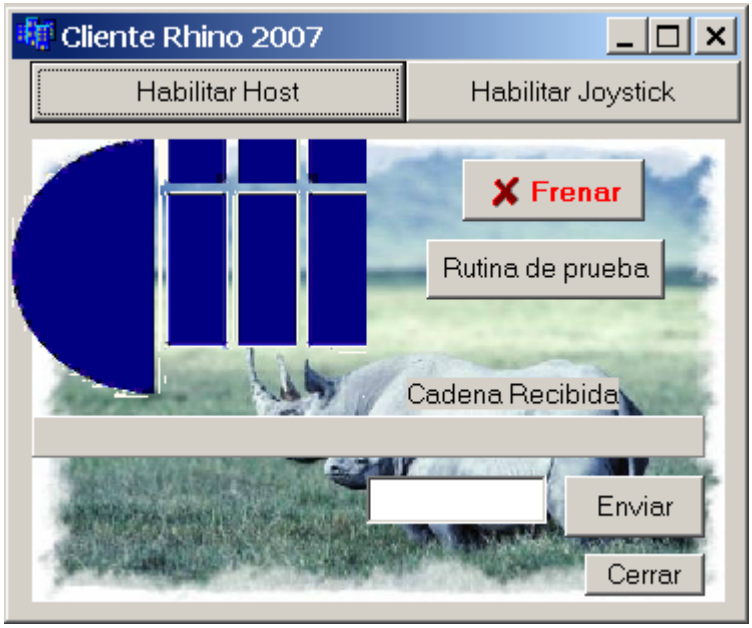
Por medio de esta aplicación es posible conectarse al servidor Rhino, que corre en la PC local del AGV, desde una maquina remota, por medio de una conexión de red. Utiliza el protocolo UDP. Para que la comunicación sea posible, es necesario configurar en el cliente la dirección IP del servidor, así como también el puerto de comunicaciones y recompilar la aplicación. Hay que asegurarse que el puerto de comunicaciones este libre a tal efecto.

A través de la aplicación cliente es posible comandar el servo robot Rhino con un un Joystick, configurándose para ello los motores en modo *Velocidad,* o por medio de una línea de comando. Cuando se trabaja en *Windows 2000* es necesario reconfigurar el controlador del Joystick desde el panel de control cada vez que arranca el sistema operativo. Esta falla no ha podido ser eliminada y causa la falla de la aplicación cliente.

La aplicación cliente cuenta también con una rutina de prueba para comprobar la comunicación con el servidor y la funcionalidad del servo

robot. Esta rutina inicializa en controlador y lo lleva a la posición de hard home. Durante este procedimiento, el grip del brazo cierra y abre una vez.
El trafico proveniente del servo robot, reenviado por el servidor Rhino, es visualizada por el cliente en su ventana. Así mismo, el servidor puede loguear mensajes propietarios en el cliente por este medio.
Para que el controlador Rhino comience a ejecutar las instrucciones enviadas por el cliente, es necesario habilitar el host, además hay que habilitar el Joystick para comenzar a utilizarlo.

2.1.1 Screenshot



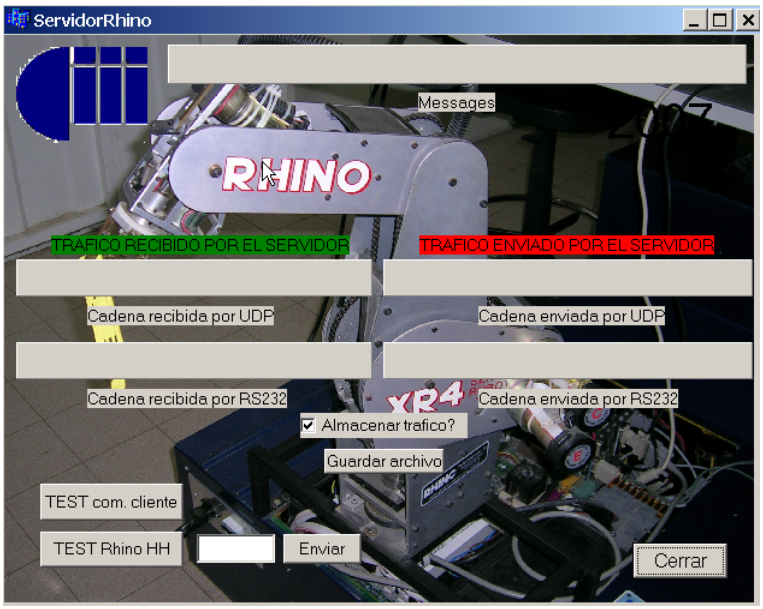
2.2 Servidor Rhino

El servidor Rhino es el encargado de reenviar el tráfico proveniente del cliente remoto hacia el controlador Rhino y viceversa, reenvía todo el tráfico proveniente del controlador Rhino hacia el cliente remoto. Cumple básicamente la función de rutear el tráfico entre cliente remoto y controlador Rhino.
Tiene la capacidad de procesar el tráfico en ambas direcciones, logueando de forma preestablecida tanto el flujo de datos, como los mensajes generados por el sistema. Para ello se vale de la clase Rhino.

Para que la comunicación con la aplicación cliente sea posible, es necesario configurar en el servidor la dirección IP del cliente remoto, así como también el puerto de comunicaciones y recompilar la aplicación. Hay que asegurarse que el puerto de comunicaciones este libre a tal efecto.

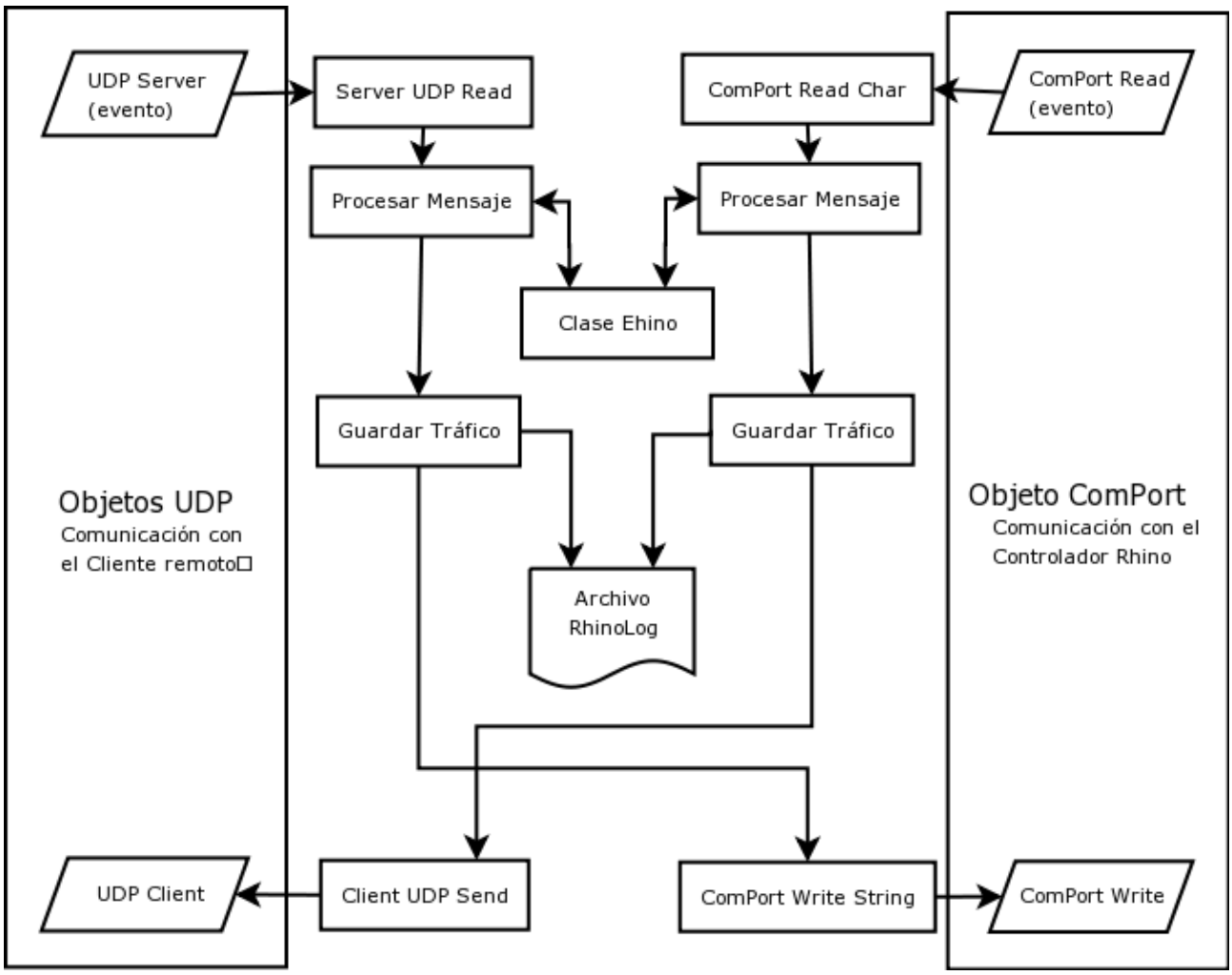
Todo el tráfico es visualizado en la pantalla del servidor, según procedencia y destino. Se visualizan mensajes generados internamente por el servidor y/o generados a partir del flujo de datos proveniente del controlador Rhino.

2.2.1 Screenshot



2.2.2 Diagrama de flujo del servidor

El servidor se comporta como un router, reenviando el trafico entre cliente y controlador. La conexión con el cliente remoto se realiza a través de un puerto de comunicaciones de red UDP. En el servidor hay que configurar la direcciona IP y puerto de comunicaciones del cliente remoto. Cuando un paquete llega al servidor desde el cliente, se genera un evento, llamándose a la función UDPServerRead, que pone el paquete a disposición del servidor. El paquete es preprocesado y reenviado al controlador Rhino por un puerto de comunicaciones serial. La función ComPortWrite, del objeto ComPort es la encargada de enviar la trama al controlador. Antes de ser enviada la trama, se pasa como parámetro a la clase Rhino y luego se almacena en disco, en el archivo *RhinoLog.txt* Cuando el controlador Rhino envía datos al server, se genera un evento, llamándose a la función ComPortReadChar, perteneciente al objeto ComPort. Esta función pone los datos a disposición del servidor una vez recibida la trama completa. Estos datos son preprocesados y reenviados al cliente remoto por medio de la función ClienUDPSend. En la etapa de preproceso, el paquete recibido es pasado a la clase Rhino para la generación de los mensajes de sistema y error correspondientes.



2.2.3 Archivos relacionados:

2.2.3.1 RhinoLog.txt

En el archivo RhinoLog.txt se almacena todo el tráfico que pasa por el servidor. Esta opción puede ser desactivada desde el panel principal.

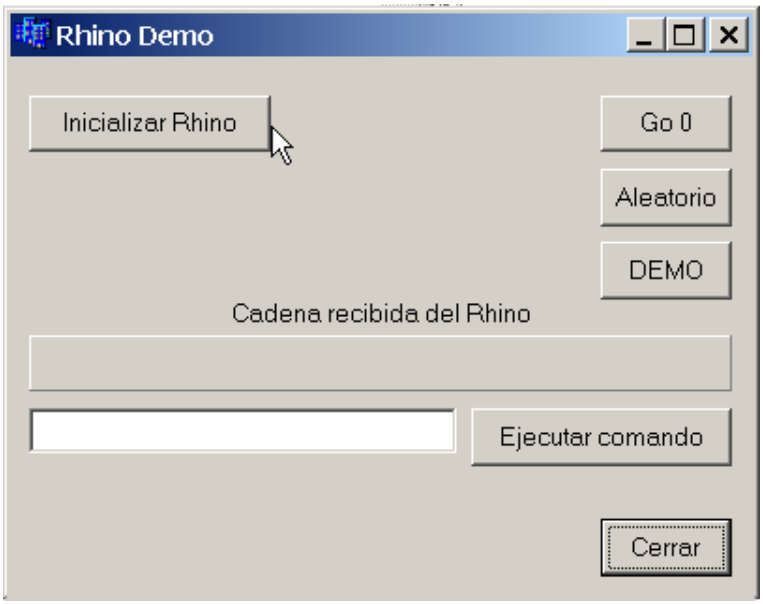
```
...
Enviado por RS232:  tx
Recibido por RS232: Copyright (C) 1993 Rhino Robotics Ltd.   V 02.00.02. SN 1993.
Enviado por UDP:    Copyright (C) 1993 Rhino Robotics Ltd.   V 02.00.02. SN 1993.
...
```

2.3 Programa Demo

Este programa puede ser utilizado para comprobar la funcionalidad del servo robot ejecutándose en forma local. Cuenta con botones para la ejecución de: una rutina de inicialización, una función para llevar el brazo a la posición de *hard home*, una función que genera movimientos coordinados de todos los motores con posición final aleatoria dentro de los margenes del sistema (elongaciones o giros

máximos) y una rutina *DEMO* con una secuencia de comandos preestablecidos, con la cual el brazo simula tomar un objeto, partiendo desde la posición de hard home, desplazándolo a otra posición, volviendo a su posición inicial y realizando la misma rutina en forma inversa. La rutina puede ser repetida para comprobar la precisión y repetibilidad del brazo.
Así mismo es posible enviar comando ingresados de forma manual, recibiendo y visualizando las respuestas del controlador en forma original, sin preproceso.

2.3.1 Screenshot



2.4 Archivos relacionados a todos los proyectos:

2.4.1 HistoricoDeCambios.txt

PROYECTO RHINO

Histórico de cambios:

Versión 1.0 Agosto 2007

- Guarda todo el trafico en un archivo
- Formatea el comando para ser enviado al controlador del Rhino (el cliente envía el comando sin formato)
- Se codifica una función de prueba para el Rhino
- Posee una función de testeo de comunicación con el cliente
- Reenvía todo el trafico proveniente de UDP hacia RS232
- Reenvía todo el trafico proveniente de RS232 hacia UDP

Cambios de la versión 2.0 Octubre 2007

- Arma paquetes completos para enviar al Cliente
- Se agrego al cliente una opción para mandar paquetes ingresados por consola
- El cliente muestra visualmente el trafico recibido

Cambios de la versión 2.1 Diciembre 2007

- Cambió la estética del cliente y el servidor
- Se incluyo la clase Rhino para el manejo de errores. La clase aun no esta completamente implementada pero es funcional (se incluyo al proyecto y compilo sin modificación alguna)
-Mediante un timer se espera que llegue la trama completa enviada por el Rhino antes de reenviarla al cliente remoto (Implementación temporal)

Cambios de la versión 2.2 Febrero 2008

- Se modifico la rutina de prueba de la clase Rhino para el manejo de errores (presenta mensajes al azar, solo ensayo)

Cambios de la versión 2.4 Febrero 2008
-Se incorporo una Versión 100% funcional, con los métodos principales codificados de la Clase Rhino.
-Ensayo sistemático del conjunto Cliente/Servidor/ClaseRhino y corrección de errores.
-Presentación visual/decodificación de los mensajes de estado y error del controlador en el servidor, reenvío al cliente.

Proyección -> Versión 3.0 Abril 2008

Se perfilaron la mayoría de los diagramas de flujo del nuevo servidor y las políticas de comunicación con el cliente y el controlador del servo robot.

- Cambios estructurales (flujo de datos)
- Implementación de la clase Rhino y mensajes de red.
- CODIFICACION pendiente.

NOTA: Al correr la aplicación cliente desde una maquina remota hay que configurar los IP, puertos y recompilar. No hay que olvidar instalar el Joystick en la maquina remota.

Viel Glueck!

Marco Alvarez Reyna
CIII UTN-FRC Argentina
marcoalrey@gmail.com

2.5 Pendientes:

Por implementar en el cliente y el servidor Rhino: Comentarios del 18/04/2008

- Modificar las aplicaciones para que sea posible cambiar las direcciones IP respectivas y los puertos de comunicaciones en forma dinámica, y no sea necesario recompilar las aplicaciones.

3 Desarrollo Clase Rhino: Gestión

Por medio de la clase Rhino es posible interpretar los mensajes de estado y de error provenientes del controlador Rhino.
La clase recibe como parámetro los comandos enviados hacia el controlador para la gestión de estado y error, y recibe también la respuesta del mismo. La clase almacena y procesa la información recibida, luego accede a arreglo de cadena de caracteres (base de datos hardcodeada) donde correlaciona estos códigos con los mensajes de sistema correspondientes, pasándolos al la capa superior, de forma previamente establecida por el nivel de logueo seteado por el usuario.
La transferencia de información de la clase al programa principal puede ser realizada por medio de funciones o por un sistema de logueo con cuatro niveles distintos de operación (se diferencian en la presentación de información al usuario)
La documentación principal de la clase Rhino fue desarrollada con Doxygen.

3.1 Utilización básica:

La secuencia básica de inicialización y utilización de la clase es la siguiente:

- Crear una instancia de la clase Rhino
- Se llama a la función *set_LogLevel()* pasando el nivel de logueo deseado como parámetro.
- Cada vez que llega un comando para el controlador se llama previamente a la función *ProcessCommandToRhino()* pasándole como parámetro el comando recibido.
- Se pasa la respuesta recibida del controlador a la función *ProcessMessageFromRhino()*
- Estos dos últimos pasos se repiten cíclicamente por cada comando recibido.
- Destruir la instancia de la clase Rhino al finalizar la aplic.

Uno de los parámetros devueltos por la función *ProcessMessageFromRhino()* es el mensaje de estado o error ya procesado, perteneciente al controlador Rhino. Es posible extraer los mensajes de sistema o error por medio de otro grupo de funciones específicas (get_*StatusMsg() get_ErrorMsg())*

3.2 Archivos de la clase:

3.2.1 rhino.cpp
Archivo de implementación de todos los métodos de la clase Rhino.

3.2.2 rhino.h
Archivo de cabecera de la clase Rhino.

3.2.3 errdef.h
Definición de todos los códigos de error del sistema Rhino.

3.2.4 cmddef.h
Enumeración de comandos del sistema Rhino.

3.2.5 statusdef.h
Definición de los principales códigos de estado del sistema Rhino.

3.3 Material extra

3.3.1 Makefile

Archivo de configuración para la aplicación *make*

```
#*********************************************************************
# Fichero Makefile
# ---------------------------------------------------------------------------------------
# Licencia GPL
# *********************************************************************
# Marco Alvarez Reyna
# CIII UTN-FRC Argentina
# Mayo 2007
# mail: marcoalrey@gmail.com
# *********************************************************************

#---- Directorios
BINDIR = .
HOMEDIR = ~

#---- Compilador
CC = g++
CFLAGS = -Wall

NAME=rhinoBot
NAME1=terminal.cpp
NAME2=rhino.cpp
NAME3=
#../serial/clase/comSerial.cpp
NAME4=rc_error_log.txt
NAME5=Doc

all:

	$(CC) -o $(BINDIR)/$(NAME) $(NAME1) $(NAME2) $(NAME3) $(CFLAGS)

doc:
	doxygen Doxyfile
	cp -rvf $(BINDIR)/$(NAME5)/html/* $(HOMEDIR)/public_html/

clean:
	rm -rvf $(BINDIR)/$(NAME) *~ *.o *# $(NAME4) $(BINDIR)/$(NAME5)/*
```

3.3.2 ChangeLog.txt

```
PROYECTO RHINO

****** Notas de creación y desarrollo de la clase Rhino ******

Histórico de cambios:

-23/05/2007 Codificación del esqueleto de la clase Rhino. Primera compilación.
            Rhino()
            ~Rhino()

-24/05/2007 Codificación de las funciones:
            ProcessErrorMsg()
            set_LogLevel()
            get_LogLevel()
            get_ErrorMsg()
            get_ErrorMsg() Sobrecargada

-26/05/2007 Codificación de las funciones:
            get_ErrorCode()
            DisplayAllErrorMessages()
            Se agrego el archivo ERRDEF.H

-29/05/2007 Se genero la primer versión de la documentación de la clase Rhino
            con Doxygen 1.3.2

-13/03/2008 Codificación de las funciones:
            ProcessStatusMsg()
            Se agrego el archivo CMDDEF.H
            Se agrego el archivo STATUSDEF.H

-25/03/2008 Codificación de funciones varias y adición de variables:
            LogMessage()
            InitMessageVectors()
            ProcessCommandtoRhino()
            ProcessCommandtoRhino() sobrecargada
            ProcessMessagesFromMessage()
            ProcessMessagesFromMessage() sobrecargada
            CBkMsg()
            get_CBkMsg()

-26/03/2008 Reordenamiento de los miembros privados y públicos.

-23/04/2008 Cierre etapa de codificación. Test global. Corrección de bugs.
            Versión 2.4 estable y testeada.

-29/05/2007 Se genero la segunda versión de la documentación de la clase Rhino
            con Doxygen 1.4.4

-07/05/2008 Comienza documentación final del Servidor/Cliente/Clase Rhino.

Marco Alvarez Reyna
CIII UTN-FRC Argentina
Inicio: Mayo 2007
mail:   marcoalrey@gmail.com
```

3.3.3 rc_error_log.txt

Formato de salida del archivo de logueo de errores de la Clase RHINO

```
...
RhinoClass->ProcessStatusMsg:
        Ultimo cmd enviado: SA
        Ultima cadena recibida: 255
        código de estado: 9
        Log Level: 3
        MSG:    Rhino Controller Status:   Motor A is executing a trapezoidal move

RhinoClass->ProcessErrorMsg:
        Ultimo cmd enviado: SE
        Ultima cadena recibida: 0
        Código de error: 0
        Log Level: 3
        MSG:    Rhino Controller Error:    BE HAPY, NO PROBLEM

RhinoClass->ProcessErrorMsg:
        Ultimo cmd enviado: SE
        Ultima cadena recibida: 12
        Código de error: 12
        Log Level: 3
        MSG:    Rhino Controller Error:    Missing parameter.
...
```

3.3.4 utilities.txt

Instructivo para realizar copias de seguridad y documentación con Doxygen de la clase Rhino.

```
PROYECTO RHINO

****** Generar copias de seguridad y documentación en LINUX ******

>cd ~/proyectos/rhino
>make clean
>rm -rvf ./Doc/*
>cd ..
>tar -czf <fecha:a.m.d>_rhino.tar ./rhino
>cd ./rhino
>doxywizard Doxyfile &    (actualizar configuración)
>doxygen Doxyfile
>cp -rvf ./Doc/html/* ~/public_html/
>make clean all            (genera el ejecutable)

La documentación estará disponible en:
usr@machine:~/proyectos/rhino/Doc/
usr@machine:~/public_html/

Viel Glueck!

Marco Alvarez Reyna
CIII UTN-FRC Argentina
marcoalrey@gmail.com
```

3.3.5 .snprj

Archivo de configuración para utilización de la aplicación *Source Navigator*.

3.3.6 Doxyfile

Archivo de configuración para utilización de la aplicación *Doxygen*.

3.3.7 Doc

Directorio de la documentación de la clase Rhino.

3.4 Programa de aplicación/testeo: terminal.cpp

Esta aplicación sirve para analizar y testear todos los casos de utilización de la clase Rhino. También sirve como ejemplo de utilización de la clase.

3.5 Pendientes:

Por implementar en la clase: Comentarios del 18/04/2008

- Cargar los mensajes de error y de estado desde un archivo (sacarlos del código fuente)
- Renombrar las variables globales bajo un mismo criterio.
- Renombrar las variables locales bajo un mismo criterio.

- Homogeneizar un idioma para los nombres de funciones.
- Implementar la función CALLBACK.
- Terminar de codificar el proceso de los comandos restantes.
- Separar en capas las distintas tareas:
  - Eliminar de la clase toda impresión en pantalla.
  - Eliminar de la clase toda escritura en HD.
- Terminar de codificar el proceso de los comandos restantes y completar la base de dato con los mensajes de sistema correspondientes.

4 **Desarrollo Clase SerialCom**: Comunicación

La clase SerialCom fue desarrollada para permitir la comunicación directa de la clase RhinoControl (aun no implementada) con el controlador del servo robot. Está basada en un conjunto de funciones pertenecientes a una librería GNU para comunicación serial, encapsuladas en una clase y con nuevos métodos.
Esta clase no es utilizada en el proyecto Rhino actualmente. Se preve su utilización una vez codificada la clase para el control directo del servo robot.
La clase fue desarrollada y ensayada bajo *SUSE Linux 9.0*, utilizando un módem externo como dispositivo de ensayo y visualización, *emacs* como editor y *gcc* como compilador.

4.1 Archivos de la clase

4.1.1 comSerial.cpp

Implementación de los métodos de la clase SerialCom.

4.1.2 comSerial.h

Archivo de cabecera de la clase SerialCom.

4.2 Material extra

4.2.1 Makefile

Archivo de configuración para la aplicación *make.*

```
#*********************************************************************
# Fichero makefile.
# -------------------------------------------------------------------
# Licencia GPL. Marco Alvarez Reyna UTN-FRC mail: marcoalrey@gmail.com
#*********************************************************************

#---- Directorios
BINDIR = .

#---- Compilador
CC = g++
CFLAGS = -Wall

NAME=apl
NAME1=terminal.cpp
NAME2=comSerial.cpp

all:

	$(CC) -o $(BINDIR)/$(NAME) $(NAME1) $(NAME2) $(CFLAGS)

clean::
	 rm -f $(BINDIR)/$(NAME) *~ *.o *#
```

4.2.2 ChangeLog.txt

```
PROYECTO RHINO

Histórico de cambios de la clase SerialCom:

Mayo 2007

Notas:
	-Se conformo la clase "SerialCom"
	-Se agregaron las funciones "getrts() rts_on() rts_off() getcts()
	getdsr()" para el manejo y gestión de las lineas de "control del módem"
	-Se agregaron comentarios dentro del código fuente
	-Se escribió el archivo Makefile necesario para compilar con "make"

Marco Alvarez Reyna
CIII UTN-FRC Argentina
marcoalrey@gmail.com
```

4.2.3 Doc

4.2.3.1 Doxygen

La documentación de la clase se genero con doxygen en formato HTNL

4.3 Programa de aplicación/testeo

4.3.1 terminal.cpp

Esta aplicación permite probar la funcionalidad de la clase.

5 Versiones futuras

5.1 Diagrama de flujo del servidor

La versión 3.0 del servidor Rhino esta en fase de desarrollo. Un 80% de los diagramas de flujo del nuevo servidor ya han sido generados, utilizando la aplicación DIA. Estos diagramas se van a adjuntar al directorio de desarrollo del servidor.

5.2 Políticas para la gestión de estado y errores

Se preve la redistribución de rutinas de emergencia entre cliente y servidor ante la aparición de un error de sistema. Para ello se establecerán niveles de ejecución de rutinas de emergencia.

6 Pendientes a nivel de proyecto global

- Diseño y codificación de la clase RhinoControl.
- Corrección de las rutinas de inicialización del controlador del servo robot.
- El brazo presenta deterioros mecánicos: engranajes flojos, falta de lubricación y suciedad. Hace falta y una revisión y service general (limpieza, ajustes, lubricación)
- Revisión y mantenimiento eléctrico. En la mano, hay fines de carrera que no están funcionando y el conector de uno de los encoders hace falso contacto, esta deteriorado. Es necesario corregir esta falla para evitar que los motores permanezcan excitados y puedan deteriorarse por exceso de temperatura.
- Implementar un mecanismo de comunicación entre aplicaciones para poder enlazar a la aplicación cliente o servidor con una fuente de datos ajena a las mismas (propuesta: MailBox)
- Establecer niveles de seguridad y acceso al servidor.
- Implementación de un gestor de procesos.
- Implementar la comunicación con el controlador Rhino por medio de un driver dedicado.
- Desarrollo de una maquina de estados portable, para ser utilizada por el usuario en la programación de rutinas de alto nivel.
- Incorporar al informe un resumen de las características del Rhino.
- Grabar un CD con todas las aplicaciones, código fuente, y documentación.

NOTAS: