Se buscan programadores para contribuir al desarrollo de herramientas de código abierto para la automatización de pruebas

Por Stefan Kopp

Stefan Kopp es Director de desarrollo comercial, Agilent Technologies

Agilent Technologies

www.agilent.com

En los últimos años, Linux se ha convertido en un tema de interés entre la comunidad de profesionales dedicados a pruebas y medidas. Mientras que los usuarios domésticos prefieren Windows® por su sencillez de uso y su aplicabilidad, algunos deciden utilizar Linux para automatizar sus pruebas. Los aficionados a Linux aprecian y promueven las ventajas de su entorno de código abierto, en particular un nivel de control, flexibilidad y estabilidad sin precedentes.

Sin duda, utilizar Linux para la automatización de pruebas no es la vía más fácil. Frecuentemente, las herramientas y los controladores comerciales no son compatibles con Linux. Y aunque lo sean, normalmente no son de código abierto y, en consecuencia, sólo funcionan con ciertas versiones y distribuciones. Por lo tanto, muchos usuarios de Linux prefieren el enfoque de "hágalo usted mismo": ¿Por qué utilizar un software binario comercial que limite su flexibilidad y que no permita aprovechar los beneficios que quería conseguir al elegir este sistema operativo? Desafortunadamente, en cierta medida, el "hágalo usted mismo" implica reinventar la rueda.

Se abre la puerta

En 2007, motivado por un aumento en las llamadas de consulta y de solicitud de asistencia recibidas, Agilent lanzó una serie de notas de aplicación sobre el control de instrumentos de prueba utilizando Linux y las herramientas de código abierto más populares. Con la esperanza de estimular el desarrollo de los usuarios, las notas iban acompañadas de extensos fragmentos de código a modo de ejemplo. La subsiguiente respuesta de los clientes y los proveedores de soluciones indicó que mucha gente estaba utilizando el código en sus aplicaciones. Algunos de ellos añadían extensiones o resolvían errores, dispuestos a compartir estas mejoras con otros desarrolladores.

Para facilitar y fomentar el desarrollo conjunto del código (siguiendo la tradición de la comunidad de código abierto), el código se ha ampliado recientemente y está disponible en SourceForge.net, un conocido sitio de proyectos de desarrollo de código abierto. Le invitamos a descargar el código, a utilizarlo y, si aplica alguna mejora, a compartirla con la comunidad.

Agilent ha bautizado su proyecto ad hoc como "openTMlib," que significa "biblioteca abierta de pruebas y medidas". El resto de este artículo describe el objetivo del proyecto, su alcance actual y la arquitectura de software propuesta.

Análisis de las directrices generales del proyecto

Cuatro ideas clave definen la dirección del proyecto: independencia de entrada/salida (E/S), independencia de lenguaje de programación, multiproceso y modularidad.

Independencia de E/S:

El código de ejemplo original mencionado anteriormente se limitaba a instrumentos basados en USB y LAN. Entrando en materia, el proyecto openTMlib tiene como objetivo crear una interfaz de programación de aplicaciones (API) común y una biblioteca más universal para todos los tipos de instrumentos basados en mensajes más conocidos. La arquitectura deberá ser compatible con la última extensión para instrumentos GPIB y serie, además de VXI (mediante un controlador Slot-0 basado en GPIB o LAN). Una API común permite al usuario alternar distintos tipos de interfaces de E/S con cambios mínimos en el código.

Independencia de lenguaje:

Aunque el código original se basaba en C, el proyecto pretende crear una biblioteca única para utilizar con diversos lenguajes de programación, incluidos lenguajes de comandos como Python. La independencia de lenguaje es un objetivo importante por dos razones: proporciona a los desarrolladores una mayor flexibilidad y hace que valga la pena invertir en controladores, aumentando la base de usuarios potenciales y aplicaciones objetivo.

Multiproceso:

Dado que la mayoría de las aplicaciones de prueba se basan en la ejecución secuencial, la adición de seguridad del proceso (p. ej. aptitud para uso en aplicaciones multiproceso o multitarea) no se considera un requisito urgente. Añadiría complejidad y aumentaría enormemente la sobrecarga de las pruebas de software. Además, en las ocasiones en las que se requiere ejecución paralela, el bloqueo y la sincronización necesarios pueden gestionarse en la capa de aplicación.

Modularidad:

Este es un objetivo importante para todo el software, pero es especialmente importante para los proyectos de código abierto en los que varios colaboradores pueden trabajar en diferentes elementos del proyecto y la coordinación es más informal que en un proyecto de software comercial. En este entorno, es esencial tener los módulos e interfaces bien definidos.

Resumen de las principales decisiones de diseño

Hay tres decisiones clave que definen la forma y dirección del proyecto openTMlib. La más importante es la decisión de utilizar la API de Virtual Instrument Software Architecture (VISA). También son importantes las decisiones de ofrecer todo el espacio de usuario que sea posible y aplicar los módulos como bibliotecas compartidas.

Acerca de SourceForge.net

SourceForge.net es un sitio Web y una plataforma de desarrollo de software de código abierto, diseñado y albergado por SourceForge, Inc., un proveedor de soluciones de gestión de proyectos de software. SourceForge.net ofrece alojamiento Web gratuito para proyectos de software no comerciales de código abierto.

API VISA:

VISA es un estándar de biblioteca de E/S mantenido por la IVI Foundation, un consorcio de las empresas más importantes de la industria de las pruebas, entre las que se encuentra Agilent. Dado que VISA es compatible con los objetivos de diseño mencionados más arriba, es una decisión natural adoptar su API siempre que sea posible. Al mismo tiempo, está más allá del alcance del proyecto openTMlib desarrollar una aplicación completa del estándar VISA, que es compatible con aplicaciones multiproceso de E/S basadas en registro y mucho más. No obstante, adoptar el API VISA permite un nivel de portabilidad hacia y desde aplicaciones Windows como las que se basan en la IO Libraries Suite de Agilent.

Espacio de usuario:

Mientras que el controlador de clase de prueba y medida USB (USBTMC) original se utilizaba completamente como módulo kernel, el proyecto openTMlib incorpora una capa de controlador adicional en el espacio de usuario. El objetivo es aplicar tanta funcionalidad como sea posible en el espacio de usuario donde es más conveniente el desarrollo, las pruebas y la depuración. La capa de controlador adicional también permite obtener una interfaz unificada para los diversos servicios de E/S

Bibliotecas compartidas:

Para aumentar la facilidad de uso y la extensibilidad de la biblioteca, los diversos módulos se aplican como bibliotecas compartidas. Esto permite actualizar y extender la biblioteca sin volver a compilar el código de aplicación y todo el código de la biblioteca.

Esquema de la arquitectura de openTMlib

En la figura 1 se muestra una descripción general de la arquitectura del software. Los elementos que se muestran en verde forman parte del alcance del proyecto, mientras que los que se muestran en amarillo forman parte del sistema operativo o de otros controladores existentes.

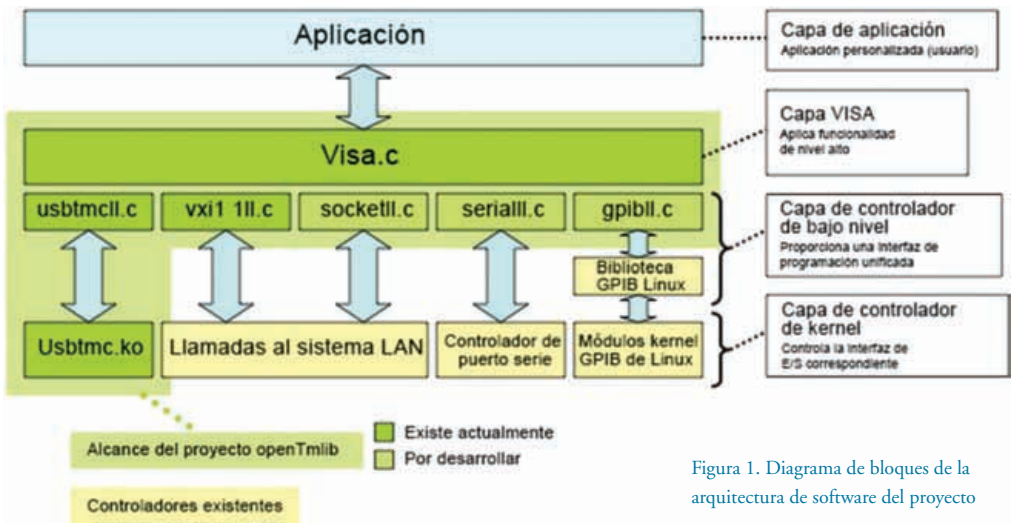

Figura 1. Diagrama de bloques de la arquitectura de software del proyecto

Función	Descripción
viOpenDefaultRM	Inicializa la biblioteca de E/S. Debe iniciarse antes de cualquier otra función de VISA.
viOpen	Abre una sesión con un instrumento.
viClose	Cierra una sesión abierta anteriormente con viOpen. También se utiliza para cerrar la biblioteca cuando se utiliza con viOpenDefaultRM.
viWrite	Envía datos (como una cadena de comandos) a un instrumento.
viRead	Lee los datos de respuesta de un instrumento.
viGetAttribute	Establece un atributo de instrumento o de sesión.
viSetAttribute	Lee el estado actual de un atributo de instrumento o de sesión.
viStatusDesc	Devuelve una cadena descriptiva (nombre) para un código de estado dado.
viClear	Limpia el búfer de E/S de un dispositivo.
viReadSTB	Devuelve el valor de bytes de estado del dispositivo.
viAssertTrigger	Activa el dispositivo utilizando una línea de disparo de hardware o un mecanismo de software parecido.
viReadToFile	Lee datos de un dispositivo y los escribe directamente en un archivo.
viWriteFromFile	Lee datos de un archivo y los envía directamente a un dispositivo.
viUsbControlOut	Se utiliza para controlar directamente un dispositivo USB (solicitud de salida de control).
viUsbControlIn	Se utiliza para controlar directamente un dispositivo USB (solicitud de entrada de control).

Tabla 1. Conjunto actual de funciones VISA aplicadas

```
#include "visa.h"

int main()
{
        ViSession dmm;
        ViSession master;
        int retval;
        ViUInt32 count;
        char buffer[2000];

        /* Initialize and open I/O library */
        if((retval=viOpenDefaultRM(&master))!=VI_SUCCESS) {
                printf("viOpenDefaultRM returned error %d\n",retval);
                return(1);
        }

        /* Open instrument session */
        if((retval=viOpen(master,"TCPIP0::15.138.4.3::inst0::INSTR",
                0,5000,&dmm))!=VI_SUCCESS) {
                printf("viOpen returned error %d\n",retval);
                return(1);
        }

        /* Request and read back instrument ID string */
        /* Further error checking omitted for clarity */
        viWrite(dmm,(unsigned char *)"*IDN?\n",6,&count);
        viRead(dmm,(unsigned char *)buffer,2000,&count);
        buffer[count]=0; /* Add end-of-string character */
        printf("Response: %s\n",buffer);

        viClose(dmm); /* Close instrument session */
        viClose(master); /* Close I/O library */
        return 0;

}
```

Figura 2. Ejemplo sencillo de control de instrumentos utilizando VISA

La capa superior de la biblioteca es la capa VISA. Incluye la funcionalidad genérica de E/S, independiente de la interfaz utilizada. Actualmente aplica las funciones de VISA que aparecen en la tabla 1. Para aquellos que no estén familiarizados con VISA, en la figura 2 se muestra un ejemplo sencillo de cómo utilizar algunas de estas funciones para comunicarse con los instrumentos.

Función	Descripción
ll_initialize	Inicializa el módulo de controlador. Debe iniciarse antes de cualquier otra función.
ll_cleanup	Lanza los recursos asignados por ll_initialize. Debe iniciarse cuando ya no se utilice el controlador.
ll_open	Inicializa una sesión de instrumento correspondiente a una cadena de recursos determinada. Utiliza la misma cadena de recursos que VISA (pasados por la capa VISA).
ll_close	Cierra una sesión de instrumento abierta anteriormente con ll_open.
ll_read	Lee datos de un dispositivo.
ll_write	Escribe datos en un dispositivo.
ll_get_attribute	Establece un atributo de instrumento o de sesión. Utiliza nombres de atributo y valores VISA (pasados por la capa VISA).
ll_set_attribute	Lee el estado actual de un atributo de instrumento o de sesión.
ll_io_operation	Realiza una operación especial de E/S, como Reinicializar, Restablecer Dispositivo, etc.
ll_report_devices	Devuelve información acerca de los dispositivos conectados a la interfaz controlada por el controlador. Esto resulta útil para los dispositivos como USB donde hay información del dispositivo disponible a través de los servicios centrales USB del kernel.

La capa siguiente es la capa de controlador de bajo nivel, que incluye un módulo de controlador separado para cada protocolo de E/S compatible (actualmente, USB y VXI-11). Estos controladores de bajo nivel sólo aplican la funcionalidad de E/S más básica, mientras que la funcionalidad más alta, como el procesamiento de datos, es común para todas las interfaces de E/S y por lo tanto se aplica en la capa VISA. El objetivo de los controladores de bajo nivel es envolver las llamadas al sistema específicas del protocolo en una API unificada para el acceso mediante la capa VISA. Las funciones que deben aplicarse se describen en la tabla 2. Tenga en cuenta que el "ll_" en el nombre de cada función corresponde a dos eles minúsculas, que significan bajo nivel ("low level").

En las capas de bajo nivel y de controlador del kernel, las aplicaciones difieren considerablemente entre las interfaces. Para instrumentos LAN y serie, Linux ofrece varias llamadas al sistema que permiten un acceso controlado desde el espacio de usuario. El acceso es bastante sencillo. Para obtener más información sobre el control de instrumentos basados en LAN, consulte las notas de aplicación de Agilent 1465-28, Using Linux to Control LXI Instruments Through VXI-11 (Uso de Linux para controlar instrumentos LXI mediante VXI-11), y 1465-29, Using Linux to Control LXI Instruments Through TCP (Uso de Linux para controlar instrumentos LXI mediante TCP).[1,2]

Los servicios centrales USB del kernel no son accesibles desde el espacio de usuario. Por lo tanto, debe aplicarse el controlador de bajo nivel para USB, al menos parcialmente, en el modo kernel. En el caso de este proyecto, la funcionalidad del controlador está dividida en el recubrimiento unificado en el espacio de usuario y el controlador principal bajo la forma de un controlador de dispositivo de caracteres USBTMC personalizado. Para obtener más información sobre USBTMC, consulte la nota de aplicación de Agilent 1465-30, Using Linux to Control USB Instruments (Uso de Linux para controlar instrumentos USB).[3]

Para GPIB, hay un controlador de código abierto disponible en SourceForge.net (hay que buscar "Linux GPIB"). Este proyecto existe desde hace bastantes años y funciona con numerosas interfaces GPIB (incluidas las de Agilent). Tal y como sucede con el controlador USB descrito más arriba, este controlador está dividido en un espacio de kernel y un espacio de usuario. Utiliza una API personalizada y, en el contexto de este proyecto, requiere un recubrimiento para ofrecer su funcionalidad de una forma unificada que sea compatible con los otros controladores de bajo nivel.

Conclusión

Si está utilizando Linux para la automatización de pruebas, le animamos a participar en el proyecto openTMlib. Para examinar de cerca el estado actual de los controladores, descargar el código, contribuir con sus mejoras o simplemente dejar sus comentarios, visite la página del proyecto en sourceforge.net/projects/linux-gpib/.

Todas las contribuciones de código son bienvenidas. Estos son algunos ejemplos de mejoras e incorporaciones que resultarían especialmente útiles:

- Incorporación de una función VISA como las que se utilizan para E/S formateada
- Controladores de bajo nivel para instrumentos serie, GPIB o LAN (no VXI-11) basados en sockets
- Aumento de la seguridad del proceso de los controladores
- Informes de errores o, todavía mejor, depuración de errores
- Instalación de RPM y paquetes para los controladores
- Documentación Ω

Referencias

Nota de aplicación de Agilent 1465-28, Using Linux to Control LXI Instruments Through VXI-11 (Uso de Linux para controlar instrumentos LXI mediante VXI-11). Número de publicación 5989-6716EN disponible en www.agilent.com/find/linux.

Nota de aplicación de Agilent 1465-29, Using Linux to Control LXI Instruments Through TCP (Uso de Linux para controlar instrumentos LXI mediante TCP). Número de publicación 5989-6717EN disponible en www.agilent.com/find/linux.

Nota de aplicación de Agilent 1465-30, Using Linux to Control USB Instruments (Uso de Linux para controlar instrumentos USB). Número de publicación 5989-6718EN disponible en www.agilent.com/find/linux.

Reconocimiento

Windows es una marca comercial de Microsoft Corporation registrada en EE. UU.