Adaptación de Gxsm: aplicación del software libre para Microscopía Túnel (STM)

Farid El Gabaly y Juan de la Figuera

6 de septiembre de 2003

1. Introducción

La microscopía de efecto túnel (STM, de las siglas en inglés Scanning Tunneling Microscopy)[**?**] forma parte de un conjunto de microscopías desarrolladas recientemente (SPM, Scanning Probe Microscopies). Estas técnicas están revolucionando el modo en que podemos observar y modificar el mundo a escala de nanómetros, y están dando un significado real al tan abusado término de "Nanotecnología". Aplicaciones espectaculares de esta técnica han sido experimentos donde se mueven átomos o moléculas de uno en uno por la superficie de un metal, incluso organizándolos como las fichas de un dominó para realizar operaciones lógicas[**?**].

El funcionamiento de un microscopio SPM (sea STM, AFM (microscópio de fuerzas atómicas), etc) se basa en un aparato capaz de mover con gran precisión un sensor cerca de una superficie, hasta centésimas de Ångstrom en el caso de STM. Para realizar dichos desplazamientos se emplean materiales piezoeléctricos (que se estiran o contraen al aplicarles un campo eléctrico) a los que se aplican voltajes del orden de unos cientos de voltios. El sensor, una punta metálica muy afilada que termina en unos pocos átomos en el caso del STM, se acerca a la superficie hasta que detecta la señal correspondiente (una corriente eléctrica entre la punta y la superficie en STM). Por medio de un sistema de realimentación, se intenta mantener constante dicha señal acercando o alejando el sensor a la superficie (Figura **??**). Si en estas condiciones se mueve el sensor (se "barre") paralelo a la superficie, las correcciones que el sistema de realimentación debe

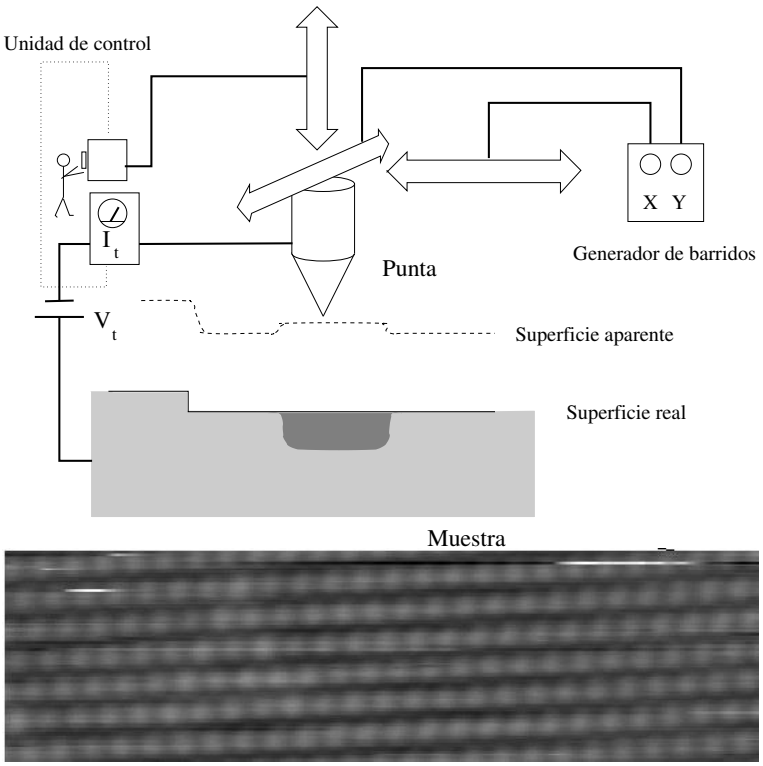

Figura 1: Panel superior: Esquema del funcionamiento del STM. Panel inferior: Imagen con resolución atómica de la superficie de un metal

realizar son una medida de la topografía de la superficie: el sensor actúa como un palo que nos indica donde están los bultos de la superficie (como ejemplo, en la figura **??** abajo, se ve la superficie de un metal, donde los bultos corresponden a átomos).

Un STM consta por tanto de una parte mecánica con posicionadores piezoeléctricos y de una electrónica de control. Dicha electrónica puede estar a su vez conectada con un ordenador para poder tratar los datos (matrices bidimensionales que mapean la altura de la superficie en cada punto). El ordenador permite dibujar los datos en falso color, o en perspectiva en 3 dimensiones, y realizar todo tipo de tratamiento de datos (filtrado, detección de objetos, etc). En algunos casos, la generación de voltajes para mover el microscopio a cada punto de la superficie, y la realimentación de control se ejecutan en la electrónica de control de forma analógica, o en procesadores DSP (Digital Signal Processor) dedicados bajo control de software.

2. El software de STM

Desde los comienzos de la era del STM (más o menos las dos primeras décadas desde su desarrollo en 1982) han proliferado los equipos desarrollados por laboratorios de investigación, con frecuencia con software desarrollado in-situ, a veces con diversos elementos comerciales. Este ha sido el caso del sistema

desarrollado en parte por uno de los autores dentro del Laboratorio de Superficies de la Universidad Autónoma de Madrid, donde se desarrolló a principios de los 90 un microscopio "casero" con una electrónica de control comercial (RHK modelo STM-100[**?**]) y un programa de adquisición también casero stm100 corriendo bajo MS-DOS[**?**] (el cual ha sido empleado en varios sistemas experimentales y dado lugar a decenas de publicaciones en revistas internacionales).

En la actualidad se pueden escoger sistemas comerciales de una gama de fabricantes que generalmente ofrecen soluciones de software y hardware conjuntamente (software, electrónicas de control, y microscopios)[**?**, **?**, **?**]. Incluso existe un software libre para MacOS 9 para el tratamiento de las imágenes de STM (Image SXM[**?**]). Asimismo se puede encontrar un software cerrado gratuito (WSxM[**?**]) que sólo funciona con su propia electrónica de control, pero que se puede emplear para el tratamiento de datos de ficheros de otras compañías. Sin embargo, no existía ningún software en código abierto para *la adquisición y el tratamiento de datos* de STM (y SPM en general). Además, la prevalencia de programas cerrados forzaba habitualmente al uso de sistemas operativos también cerrados (todos los programas comerciales actuales que conocen los autores sólo funcionan en sistemas operativos de Microsoft, y en varios casos sólo en versiones obsoletas). Todo esto resulta sorprendente dada la gran cantidad de códigos desarrollados con dinero público en Universidades e Institutos de investigación. Generalmente dichos programas, o bien han pasado a ser propiedad de empresas comerciales, o bien languidecen por falta de masa crítica de desarrolladores y el natural relevo de investigadores pre y postdoctorales que son los que usualmente realizan dicho desarrollo. De hecho ambos destinos (pasar a ser propiedad de una empresa, y no ser mantenidos) no son excluyentes.

Uno de los grupos que había desarrollado un software propio decidió hace unos pocos años dar el salto de abrir dicho software bajo licencia GPL creando el Gxsm[**?**]. Basado en la librería GTK+[**?**], y con una moderna arquitectura de plug-ins para la mayor parte de su funcionalidad, no es sólo un proyecto de software sino que ofrece una implementación de referencia también para la electrónica de control, basada en un DSP. Está alojado en SourceForge[**?**] y cuenta con un equipo de desarrolladores (como es habitual en el software libre) de distintos países[**?**].

En el laboratorio de los autores se está poniendo a funcionar un STM casero. Se disponía de una electrónica de control RHK. Antes que emplear el vetusto programa stm100 mencionado anteriormente se decidió utilizar el programa Gxsm, modificándolo para poder comunicarse y adquirir los datos de dicha electrónica de control. Esta ponencia es una descripción de los problemas y soluciones encontrados en esta labor.

Hay que destacar que muchas veces no es fácil obtener la información técnica necesaria para controlar mediante un programa externo una electrónica de STM. Muchas compañías se niegan a proporcionar esquemas y diagramas de conexión (especialmente aquellas compañias que ofrecen sistemas completos). La electrónica STM-100 de RHK venía con esquemas de circuitos y la información necesaria para controlarla desde un programa casero, aunque aparentemente la política de la compañia RHK en este aspecto ha cambiado desde entonces.

La solución empleada consistió en separar la adquisición y el interfaz directo entre la tarjeta digitalizadora que está conectada físicamente con la electrónica RHK STM-100 del programa Gxsm. Un programa intermedio, rhk_controller, es el encargado de recibir órdenes de adquisición de datos y hablar con el hardware

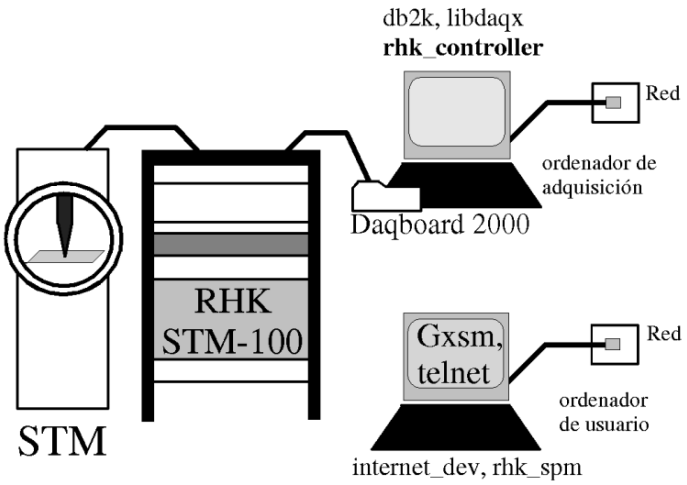

Figura 2: Esquema de las distintas partes de software y hardware de STM. La conexión de red está basada en sockets TCP

(a través de una librería y un device driver). Para hablar con el programa se emplea una conexión por sockets. La flexibilidad de esta solución es que permite manejar el microscopio incluso desde una sesión telnet. Después se modificó el programa Gxsm para poder hablar con rhk_controller. El esquema de bloques del conjunto se puede ver en la figura **??**. Discutiremos los distintos elementos, empezando por el hardware de adquisición.

3. El hardware de adquisición y su device driver

El hardware de adquisición de datos empleada es una tarjeta digitalizadora IOtech [**?**]Daqboard/2000. Las principales características de esta tarjeta PCI son:

- ocho canales de entrada analógica de 200kHz y 16 bits.
- dos canales de salida analógicas (DAC) de 100kHz y 16 bits.
- 40 entradas/salidas digitales TTL.
- DMA bus mastering
- dos contadores

La principal condición que limitaba la elección (aparte de disponer de software en GNU/Linux) era que la tarjeta seleccionada debía tener al menos 8 entradas digitales y 8 salidas también digitales además de los canales analógicos para manejar la electrónica RHK. Esto eliminaba las tarjetas de National Instruments las cuales se hayan relativamente bien soportadas bajo GNU/Linux (al menos ciertos modelos[**?**]).

La tarjeta está también soportada mediante comedi[**?**]. Sin embargo, dicho driver no ha sido actualizado recientemente, y ha sido realizado sin información

directa por parte de IOtech a partir de los drivers de Windows. Se decidió usar el driver proporcionado por IOtech en su página web. El soporte de software por parte de IO consta de un pequeño README y un fichero comprimido Daqboard2000.tgz con:

- El módulo del device driver, db2k
- La librería de acceso al device driver desde el espacio de usuario, libdaqx
- Ejemplos de uso de la librería

La versión utilizada del driver es la 0.1, lo cual hace suponer que puede tener algunos problemas. Desde el punto de vista positivo (y que ha resultado crucial), todo el driver está cubierto por la licencia GPL. IOtech tiene una dirección de correo dedicada a soporte linux[**?**].

La librería es la misma que se puede usar desde Windows (C,VB), y está bien documentada en forma de manuales en pdf.

Se encontraron a lo largo del desarrollo los siguientes problemas, una vez que se descubrió que el módulo db2k no funcionaba correctamente si era compilado bajo gcc-3.2:

1. La librería libdaqx *no* es segura para threads (*thread-safe*)
2. Algunas funciones de adquisición simplificada no funcionan
3. El driver llama copiosamente a `printk` para dar mensajes
4. El device driver dejaba de funcionar tras un cierto número de llamadas

El primer problema se solucionó no utilizando threads, aunque esto supone una complicación en el código del programa rhk_controller. La posibilidad de utilizar multithreading permitiría, por ejemplo, vaciar el buffer al mismo tiempo que se llena, y así no saturarlo nunca. Con multitheading también habría sido más fácil programar la aproximación del microscópio, por la cual la punta del STM se va acercando de forma progresiva a la muestra que queremos medir. Es un proceso delicado que conlleva el riesgo de chocar la punta contra la muestra. Los threads permitirían realizar el movimiento mientras se comprueba si hay señal en la punta. Sin multithreading, y por tanto sin multitarea, la única posibilidad ha sido programar un movimiento del tipo mover-comprobar-mover y que termine al recibir cualquier comando. Asimismo el uso de las funciones de adquisición simplificadas es completamente opcional, sin merma de posibilidades. Los puntos 3 y 4 han podido ser solucionados gracias a disponer del código fuente del device driver para su modificación. Los numerosos `printk` hacían muy lento la llamada repetida a cualquier rutina del device driver, pero es comprensible su presencia en código alfa.

Por el contrario, el último problema proviene de cómo reserva la memoria el device driver. Dicho device driver reserva un buffer contiguo de memoria de forma que la adquisición de datos se pueda producir a mayor velocidad que la lectura de dichos datos por el programa final. El tamaño de dicho buffer puede ser de hasta 2Mb, lo cual es especialmente importante en nuestra aplicación y aún más dado que no podemos emplear threads para vaciar el buffer simultáneamente. El problema es que, en el código original, la memoria se reservaba con `get_free_pages` y *se devolvía* y se volvía a reservar con cada nueva configuración

de canales y condiciones de adquisición. Tras cierto tiempo con el ordenador encendido la llamada a `get_free_pages` no era capaz de encontrar un bloque de memoria contiguo de 2Mb (`__get_free_pages(9)`), y devolvía error. Rebuscando en la web, encontramos esta referencia sobre el problema de Ingo Molnar, del 09/30/2001 en la lista del kernel (por cierto, lo mismo se puede leer en [**?**]): Subject: RE: `__get_free_pages()`: is the MEM really mine?

> "Being able to allocate a 2 MB page is only guaranteed during boot-up. There is just no mechanism in Linux that guarantees it for you to be able to allocate a 2 MB page, in even a moderately utilized system. Scatter-gather avoids all these problems."

La solución encontrada consistió en modificar el device driver para que reservase la memoria sólo una vez al cargar el módulo, y cargar éste el primero. Aun así no es una solución muy elegante, pero con la memoria que tiene un ordenador hoy en día (512Mb en nuestro caso) funciona correctamente. Aunque se ha enviado el patch del device driver a IOtech no se ha recibido ninguna contestación al respecto (ni sobre el diagnóstico). Claramente aunque el soporte Linux de IOtech existe (respondieron a algunos de los mensajes sobre el problema), carece de suficiente personal o interés. Queremos destacar que si el device driver no hubiese sido de código abierto, habría sido imposible solucionar el problema.

4. El programa de control rhk_controller

El programa que habla directamente con la librería y el device driver no es el Gxsm. Es un pequeño programa en C que se controla y envía sus respuestas mediante sockets TCP llamado rhk_controller. Interpreta las órdenes con un analizador sintáctico descrito en yacc y un analizador léxico descrito en lex, cuya entrada viene directamente de un socket. Ello permite extender muy fácilmente el pseudo lenguaje de control empleado. La ventaja de dicho pseudo lenguaje es que el programa se puede manejar desde una sesión telnet abierta sobre el puerto correspondiente (5027 por defecto). Hay de indicar que no hay ningún tipo de autentificación incluido en el programa, así que es responsabilidad del usuario usar un cortafuegos o permitir conexiones sólo desde localhost para evitar intrusiones.

A continuación se muestra un ejemplo de interacción con el programa por medio de telnet (donde el texto introducido por el usuario se muestra en itálicas):

```
juan@ibuk3:~$ telnet localhost 5027
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
verbose on
read_settings
AUX     0
I       1000
BIAS    2000
Z       3000
YSCAN   4000
YOFF    5000
```

```
XSCAN   6000
XOFF    7000
Line_rate 40.000000
Line_frame 512
Ad_gain 1
logout
Connection closed by foreign host.
juan@ibuk3:~$
```

Inicialmente el programa se había diseñado para ser multi-threaded, pero al descubrir que la librería correspondiente a la tarjeta de aquisición de datos no funcionaba desde distintas threads se abandonó dicha idea. La adquisición de datos por el momento descansa en la presencia de un buffer de 2Mb de datos organizado por el device driver db2k. Por ello, se pueden obtener imágenes al máximo de velocidad de conversión de la tarjeta Daqboard 2000 (200kHz) hasta de un tamaño de 512×512 pixels. Ello limita la máxima velocidad de barrido a 2ms en el caso de imágenes de 256 pixels (4ms para 512 pixels), y por tanto obtener una imagen cada 0.5s (2s para 512 pixels).

Para realizar pruebas con el programa sin necesitar tener conectada la tarjeta Daqboard 2000 al ordenador, se ha implementado un modo de emulación que genera datos que son enviados de la misma manera que los datos reales. Dicho modo de emulación se activa bien si se indica alguna opción al ejecutar el programa, o si se detectan errores al inicializar la tarjeta de adquisición y su librería libdaqx.

5. Modificaciones de Gxsm

El programa Gxsm (véase la figura **??**) ha sido escrito bajo C++ empleando el toolkit gráfico GTK+. Emplea clases para definir los distintos tipos de hardware, y la mayor parte de su funcionalidad reside en plug-ins que se pueden montar y desmontar mientras el programa está corriendo. Aunque en un futuro los distintos tipos de hardware corresponderán también a plug-ins, ahora hay que modificar el núcleo del programa. El hardware soportado antes de comenzar este proyecto incluye dos tipos distintos de DSP, que se ocupan de realizar la realimentación del microscopio así como de controlar los barridos. Además hay un soporte rudimentario para dispositivos comedi[**?**].

Para su interfaz con rhk_controller se añadieron dos nuevas clases de hardware, una muy general (`internet_dev`) que sólo incluye la comunicación bidireccional con un socket, y una (`rhk_spm`) que tiene en cuenta las peculiaridades del pseudo lenguaje del programa rhk_controller. También es responsabilidad de `rhk_spm` emular algunos comandos del programa Gxsm traduciéndolos a los correspondientes comandos del programa rhk_controller.

El programa Gxsm es capaz de funcionar con la electrónica RHK sin cambios en los plug-ins. Sin embargo, el modo de funcionamiento con la electrónica RHK es muy diferente de los otros dispositivos como los DSP: en este caso, la propia electrónica RHK realiza por si sola la realimentación, y genera los barridos. La posición del microscopio dentro del area de barrido también se controla mediante potenciómetros de la propia electrónica. Esto hace que sea deseable un nuevo plug-in que se encarge de la adquisición de datos incluyendo estas particularidades: fundamentalmente debe leer una serie de parámetros

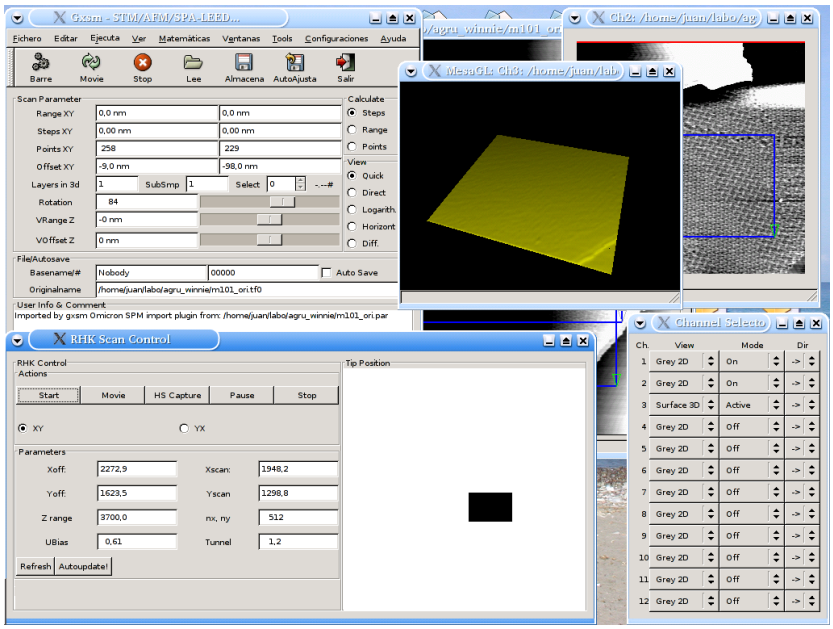

Figura 3: El programa Gxsm con el plug-in `rhk_scancontrol`

de la electrónica RHK en vez de asumir que es el propio programa quien proporciona dichos parámetros. Por ello, tomando como base de partida el plug-in `spm_scancontrol` se programó un nuevo plug-in `rhk_scancontrol` (se puede observar en la figura **??**.

6. Conclusiones

En el actual campo de la microscopía STM y dada la rápida evolución del mismo, es importante disponer de la posibilidad de modificar sin limitaciones el sistema experimental. El software Gxsm proporciona esta flexibilidad al proporcionar un sistema de adquisición de datos abierto. Además su licencia abierta lo pone a salvo de las frecuentes desapariciones de empresas (por quiebra, absorción, etc) en este sector. Hemos añadido soporte para una electrónica de STM comercial, completamente distinta en cuanto a funcionamiento del hardware para el que el programa Gxsm ha sido originalmente diseñado, empleando herramientas libres en su desarrollo.

7. Agradecimientos

Los autores agradecen el apoyo por parte de la Comunidad Autónoma de Madrid con el proyecto No. 07N/0041/2002. J. de la F. agradece al Ministerio de Ciencia y Tecnología su apoyo a través de un contrato “Ramón y Cajal”. Y por supuesto, agradecen la ayuda de la comunidad de software libre en general (el desarrollo ha tenido lugar en varios ordenadores con Debian GNU/Linux), y a los desarrolladores del Gxsm[**?**] en particular, sin cuyo esfuerzo este trabajo

no habría sido posible.

Referencias

[1] G. Binnig, H. Rohrer, C. Gerber, y E. Weibel. *Phys. Rev. Lett.*, **49** (1982) 57.

[2] A. J. Heinrich, C. P. Lutz, J. A. Gupta y D. M. Eigler, *Science* **298** (2002) 1381.

[3] RHK Technology Inc., `http://www.rhk-tech.com`

[4] J. de la Figuera, Tesis de doctorado, Universidad Autónoma de Madrid, 1996, `http://hobbes.fmc.uam.es/~juan`.

[5] Una lista no completa incluye Digital Instruments (`http://www.di.com`), Molecular Imaging (`http://www.molec.com`), Omicron Nanotechnology GmbH (`http://www.omicron-instruments.com`), y Specs Scientific Instruments, (`http://www.specs.com`).

[6] Nanotec Electrónica S.L., `http://www.nanotec.es`.

[7] S. Barret, *Proc. Royal Micros. Soc.* **37** (2002) 167, `http://reg.ssci.liv.ac.uk/`.

[8] P. Zahl, M. Bierkandt, S. Schröder, and A. Klust, *Rev. Sci. Instr.* **74** (2003) 1222.

[9] GTK+, the GIMP toolkit, `http://www.gtk.org`.

[10] Gxsm project at `SourceForge`, `http://gxsm.sourceforge.net`.

[11] La lista de desarrolladores se puede encontrar en `http://sourceforge.net/project/memberlist.php?group_id=12992`. En Junio del 2003 consta de Markus Bierkandt, Chris Angell, Thomas Schmidt, hubert, Juan de la Figuera, Andreas Klust, Martin Kneppe, Christoph Seifert, Stefan Schroeder, Thorsten Wagner y Percy Zahl.

[12] IOtech Inc., `http://www.IOtech.com/`. La dirección de correo de soporte linux es linux@IOtech.com.

[13] National Instruments, `http://www.ni.com/linux`.

[14] Comedi, The Linux Control and Measurement Interface, `http://www.comedi.org`.

[15] A. Rubini y J. Corbet, *Linux Device Drivers*, 2nd Edition, O'Reilly (1998).