Presente y futuro de eXeLearning (intef7, 01/2013)

Hitos conseguidos en esta versión

1. Desarrollo de interfaz usando el framework javascript ExtJS 4.1.

2. Reimplementación de clases Python del directorio exe/xului bajo exe/jsui para la nueva interfaz.

3. Conexión de la nueva interfaz con las nuevas clases de exe/jsui.

4. Eliminada opción xulDir del fichero de configuración.

    Añadida opción jsDir (en realidad se eliminado todo el directorio exe/xului y reubicado lo que todavía sigue siendo necesario).

5. Implementado *PackageStore* por sesión en lugar de por instancia de aplicación.

    Diferentes sesiones del navegador no comparten paquetes aunque tengan la misma url. eXe estaba pensado para ser mono-sesión (y por tanto mono-usuario). Esto es un escollo importante para tener una aplicación web. Hay tres componentes cruciales y que hay que reubicar en eXe para tener finalmente un servicio web al uso: el almacén de paquetes (*PackageStore*), el almacen de Idevices (*IdeviceStore*) y la configuración.

    En versiones anteriores, dichos componentes dependen o cuelgan de la instancia de aplicación (son globales). Con este hito se ha conseguido que el *PackageStore* sea por sesión, no por aplicación. Cada acceso desde un navegador web a la aplicación web es una sesión. Por lo tanto, ahora ya tenemos una aplicación web que pueden usar varios usuarios, cada uno con su almacén de paquetes. Sin embargo todavía comparten configuración e Idevices. Por ejemplo, si un usuario cambia el idioma, se cambiará para todos, si un usuario añade un Idevice se añadirá para todos, etc. Además, todos ven el sistema de archivos del servidor tal y como lo ve el usuario que lanzó eXe (con sus privilegios).

6. Preparado el sistema de traducción por petición del navegador (*request*) en lugar de globalmente.

    Este hito es un pequeño paso para conseguir trasladar algo de configuración a la sesión. Una sesión es un conjunto de peticiones (*requests*) desde el navegador. Ahora podemos decidir el idioma por petición: las peticiones que se hagan desde una sesión podrán tener un lenguaje diferente que las de otra sesión.

    La idea es que se siga pudiendo cambiar explícitamente el idioma desde eXe (y almacenarlo en la sesión, no globalmente).

7. Desarrollo de *filepicker* con autocompletado y atajos de teclado bajo ExtJS 4.1 para suplantar al *filepicker* XUL.

    El *filepicker* implementado debe ser ejecutado en el cliente pero debía seguir siendo remoto (consultar ficheros del servidor). Pese a las limitaciones que esto implica (el *filepicker* de un sistema operativo puede ser muy potente), el desarrollado es suficientemente ágil, aunque no será tan bueno como el nativo de un sistema operativo a menos que se invierta mucho esfuerzo.

8. Migración del toolkit de traducción a Pybabel (para extraer cadenas a traducir de archivos javascript de la nueva interfaz).

9. Se lanza eXe con el navegador por defecto del usuario.

    Ya no estamos obligados a usar Firefox. Tampoco se incluirá Firefox en los empaquetados para Mac y Windows.

10. Posibilidad de abrir múltiples ventanas/pestañas sobre un mismo paquete. Sincronización de dichas pestañas/ventanas cuando se realizan acciones.

    eXe no estaba preparado para trabajar con múltiples pestañas. Esta es una funcionalidad que necesita un gran número de pruebas.

11. Implementado mecanismo para prevenir el cierre accidental de la ventana o pestaña del navegador.

    Se termina el proceso exe solamente cuando se cierra la última ventana/pestaña (cliente) activa.

12. Aplicados parches para que eXe funcione en Python 2.7 en Windows y Mac.

13. Implementación de atajos de teclado y navegación de menús equivalentes a interfaz XUL.

    El número de atajos es limitado, por la propia complejidad del uso de atajos de teclado en javascript y posibles incompatibilidades.

14. Limpieza de código JS y HTML, adecuación a estándares y mejoras de accesibilidad en las páginas exportadas.

Posibles mejoras o caminos a seguir

1. Convertir a eXe en un verdadero servicio web.

    Actualmente ya es una aplicación web, pero es necesario implementar el objeto usuario en eXe.

    Dicho objeto albergaría el *PackageStore*, el *IdeviceStore*, la configuración de usuario y el almacenamiento privado del usuario.

    Para ello primero sería necesario un mecanismo de autenticación en el cual se asociarían sesiones a usuarios. El usuario podría tener múltiples sesiones abiertas en diferentes navegadores o ubicaciones, compartiendo en ellas sus paquetes, idevices, configuración y almacenamiento (pero no con otros usuarios).

    Dando un paso más, selectivamente podría compartir algo de su almacenamiento o paquetes (para editarlos colaborativamente en tiempo real) con otros usuarios.

    En algunos hitos conseguidos ya se ha adelantado, pero principalmente hace falta el objeto usuario, autenticación y que los 4 componentes mencionados pasen a pertenecer al objeto usuario.

2. Selector de archivos.

    Se mantendría el *filepicker* remoto actual, pero que solamente pudiera explorar el almacenamiento del usuario autenticado. Para ello habrá que implementar el objeto usuario además una capa de seguridad. Por ejemplo el servicio web deberá de ejecutarse en una jaula *chroot* para que los usuarios no puedan explorar todo el sistema de archivos del servidor con los privilegios del servicio web. Si además optamos por la compartición habrá que implementar en el *filepicker* el sistema de permisos. Se implementaría en el *filepicker* una sección para subir archivos locales, con la idea de poder subir archivos de diferentes fuentes: Google Drive, Dropbox, etc.

    Otra opción es usar un *filepicker* local que igualmente implementara las posibilidades de coger archivos de diferentes fuentes. Pero esto cambiaría la filosofía de eXe, en la que siempre han usado el *filepicker* remoto. Además el cambio de filosofía implica cambiar mucho código (mecanismos de subida y bajada de archivos, entre otros).

    Menos costosa sería la primera opción, pero para los usuarios del servicio web significa subir primero recursos y luego usarlos, aunque se haga desde la misma ventana. Implica tener un almacenamiento más en otro servicio.

    En cualquiera de las 2 opciones hay que tener en cuenta que la subida de archivos con las conexiones actuales puede ser tediosa. Esto es insalvable en el caso de subida local. Pero en el caso de servicios web de almacenamiento como Google Drive o Dropbox podría haber una solución: subir las credenciales de autenticación de estos servicios al servicio web eXe. De esta forma la subida se haría desde el servicio web, no desde el equipo del usuario, y se supone que con mayor ancho de banda. El problema es que dar permiso a la aplicación X para acceder a tu cuenta de Y podría intimidar a algunos usuarios, aunque liberar el servicio con AGPL y una buena política de servicio convencería a la mayoría.

    Esto sería necesario si queremos tener un servicio web: algo que podamos ejecutar en un servidor al lado de una Moodle, por ejemplo, que sería el objetivo final. Paralelamente se podría mantener lo que ya tenemos: una aplicación web multisesión y por tanto multiusuario, aunque los privilegios, configuración e Idevices sean los del usuario que lanzó el proceso.

3. Importación y edición de SCORM.

Para mejorar el nivel de interoperabilidad de eXeLearning, se desarrollaría una herramienta de importación de SCORM, que permitiría a los usuarios de la aplicación abrir y editar los paquetes en eXe.

Actualmente, la aplicación exporta en varios formatos estándar. Mejorando las opciones de importación, eXe podría convertirse en una herramienta de autor capaz de manejar archivos descargados de repositorios como Agrega.

Estos archivos, una vez modificados, podrían ser exportados para utilizarlos en diferentes plataformas. Esta sería una gran contribución al intercambio de información y contenidos educativos.