Anexo E. Extensión del sistema

En este anexo se presentan tres escenarios de extensión de las funcionalidades del sistema VELOAT, que se considera que es posible que surjan a futuro. Estos escenarios son: la adición de una nueva categoría del estándar LOM, la adición de funcionalidades en las que no se recomienda el uso de *managers*, y la creación de *managers* independientes del sistema desarrollado para este proyecto de tesis. A continuación se presentan guías de acción para cada escenario.

Adición de una nueva categoría de LOM

En este escenario, se estaría haciendo una extensión al estándar de metadatos LOM agregando una nueva categoría adicional a las 9 existentes, de multiplicidad permitida 1. Para empezar, es necesario primero modificar los *schemas* del estándar y regenerar el archivo *ScormCam.jar*, como se indica en la sección 4.2.2. Una vez hecho esto, los pasos a seguir para implementar la extensión del estándar en el sistema son los siguientes:

1. Crear una nueva vista nueva vista, que sea subclase de *JPanel:*
    a. Agregar métodos *get* públicos para la información que pueda ser enviada al controlador y que pueda actualizar el modelo, tales como textos en *TextFields.*
    b. Implementar la interfaz *MetadataView* del paquete *framework.*
    c. Definir los métodos de la interfaz *MetadataView:*

public void linkToController(Controller c){}

public void update(Observable subject, Object parameters){}

public void refreshView(XmlObject object){}

2. Agregar la nueva vista a la clase *LomView:*
    a. Declarar un atributo privado del tipo de la nueva vista.
    b. Inicializar la vista.
    c. Agregar la nueva vista al *TabbedPane* de las categorías de LOM.
    d. Definir un método *get* público para la nueva vista.

3. Crear un nuevo modelo, que sea subclase de *MetadataModel:*
    a. Declarar un atributo privado correspondiente al *XMLBean* principal del modelo.
    b. Definir métodos *get* y *set* públicos para el *bean* principal.
    c. Sobrescribir el método *public void generateMetadata(Object content, Object context)* si se desea hacer generación de metadatos sobre la nueva categoría.
    d. Definir las constantes y métodos correspondientes a la actualización del modelo, así como la notificación a las vistas utilizando el método *notifyObservers(Object parameter)*, heredado de la clase *Observable*.

4. Crear un nuevo controlador, que sea subclase de *MetadataController*
    a. Definir métodos *get* privados para los atributos *Model* y *View* de la clase, y que hagan el *casting* de los tipos *framework* a los subtipos nuevos.

b. Sobrescribir si es necesario el método *public void applyChanges()*, de manera que obtenga de la vista información a través de métodos *get* y se la envíe al modelo para actualizarlo.

c. Implementar el control de eventos que sea necesario

5. Crear un nuevo *manager*, que sea subclase de *Manager*

   a. Definir el método *protected void initComponents()*, que inicialice el atributo *Controller* de esta clase con construyendo una nueva instancia del *controller* recién creado.

   b. Definir métodos *get* públicos para los atributos modelo, vista y controlador de la clase, que hagan el *casting* de los tipos *framework* a los subtipos recién creados

   c. Sobrescribir sólo el constructor con firma (*Model, View*)

      i. en el cuerpo del constructor, llamar primero a *super(Model,View)*

      ii. después, llamar a *getView().refreshView(getModel().getXmlObject))*, donde *getXmlObject*() es el método *get* del *bean* principal en el modelo

6. Agregar el nuevo modelo a *LomModel*

   a. Declarar un atributo privado y un método *get* público para el nuevo modelo.

   b. En *initModels()*, inicializar el atributo, creando una nueva instancia y usando el método *set* del *bean* principal.

   c. En el cuerpo de *generateMetadata()* de *LomModel*, llamar a *generateMetadata()* del atributo del modelo recién creado.

7. Agregar el nuevo controlador a *LomController:*
    a. Declarar un atributo privado del tipo del controlador creado.
    b. Definir método *set* público para el controlador creado.
    c. En el cuerpo del método *applyChanges()* de *LomController*, llamar a *applyChanges()* del controlador creado.

8. Agregar el *manager* creado a *LomManager:*
    a. Declarar un atributo privado del tipo del *manager* creado
    b. En *initComponents*() de *LomManager*, construir el atributo *manager* agregado
    c. En *initComponents()* de *LomManager*, llamar al método *getController().setXmlObjectController(XmlObjectManager.getController()),* donde la frase "*XmlObject*" se sustituya por el nombre del *controller* y el *manager* recién creados.

9. Finalmente, se tiene el esqueleto del nuevo *manager*, sólo resta terminar de codificar los métodos creados y agregar más métodos, atributos y/o *managers* según sea necesario. Se sugiere seguir el siguiente orden:
    a. Codificar *refreshView()* en la vista creada.
    b. Codificar *linkToController()* en la vista recién creada.
    c. Implementar el manejo de eventos en el controlador recién creado.
    d. Implementar la actualización del modelo y la notificación a las vistas necesarias.

Adición de un nuevo diálogo para captura de datos del usuario

En este escenario, se requiere de un nuevo componente en el sistema, que aparentemente es candidato a ser un *manager*, pero en caso de implementarlo como tal, sería bastante sencillo. En este caso es preferible evitar el uso de *managers*, ya que su uso excesivo tiene como consecuencia un aumento excesivo también en el número de clases del sistema, y por lo tanto en el tiempo de desarrollo. Como se pudo observar en la sección anterior, el procedimiento no es tan sencillo y sólo debe realizarse cuando convenga separar en módulos una funcionalidad muy compleja. En esta situación es mejor desarrollar un solo componente que haga las funciones de modelo, vista, controlador y *manager* a la vez. VELOAT cuenta con un *framework* que facilita esta tarea, como se muestra en seguida.

1. Crear el panel de contenido del nuevo diálogo, que sea subclase de *DialogContentPane*, del paquete *framework:*
    a. Sobrescribir, si es necesario, los siguientes métodos:

    public void addDialogKeyListener(KeyAdapter okKeyAdapter),

    en caso de querer agregar *KeyListeners* a los componentes que se desee que cierren el diálogo al presionar *enter* sobre ellos

    public void addIdKeyListener(KeyAdapter idKeyAdapter),

    en caso de querer agregar *KeyListeners* a campos de texto en donde no se deseen entradas con espacios en blanco

    protected boolean hasEmptyRequiredFields(),

    en caso de contar con campos requeridos, para indicar si están vacíos

b. Agregar componentes gráficos de interacción y optar por alguna de las siguientes opciones:
    i. definir métodos *get* para todos los datos obtenibles desde este panel
    ii. definir una clase *bean* que encapsule la información obtenible y hacer un solo método get que regrese un objeto de esa clase

2. Usar la clase *BaseDialog* para mostrar el diálogo:
    a. Crear una nueva instancia de *BaseDialog* y del panel de contenido recién creado
    b. Establecer el panel de contenido del diálogo, con el método *setDialogContentPane(DialogContentPane p)*, pasando como atributo la instancia del panel recién creado.
    c. Hacer visible el diálogo.
    d. Manejar el resultado del usuario haciendo uso del método *getUserOption()* del objeto diálogo, para saber si el usuario aceptó o canceló. Si aceptó, obtener la información ingresada por el mismo usando los métodos *get* del panel de contenido recién creado.

El siguiente es un ejemplo de código para el manejo de VCards en LOM:

```
VCardEditor editor = new VCardEditor(); // extends DialogContentPane
BaseDialog dlg = new BaseDialog();
dlg.setTitle("Nuevo autor");
dlg.setDialogContentPane(editor);
dlg.setVisible(true);
if(dlg.getUserOption()==BaseDialog.OK)
      getModel().addEntity(editor.getVCardBean().toString());
```

Creación de componentes modulares e independientes

Este escenario es el más complicado de los tres abordados, porque es una mezcla de los dos anteriores. Sin embargo vale la pena mencionarlo debido a los resultados que se obtienen.

Generalmente este escenario sucede cuando se necesitan agregar nuevas funcionalidades que son extensas, requieren de modularidad, y por lo tanto el uso de *managers* es recomendado, pero también es necesaria o deseable la cualidad de independencia del módulo central del sistema, como es el caso de *LomManager* y *VoiceRecordingManager*.

Los siguientes pasos son para crear nuevos componentes con estas características y que se presenten al usuario como diálogos, y para el desarrollador constituyen ya sea parte del sistema y/o componentes independientes y reutilizables en otros sistemas.

1. Seguir el paso 1 del primer escenario, pero con los siguientes cambios:
    a. Que la nueva vista sea subclase de *DialogContentPane* y a la vez que implemente alguna de las interfaces *View* o *MetadataView*.
    b. Sobrescribir los métodos de *DialogContentPane* que sean necesarios:

        public void addDialogKeyListener(KeyAdapter okKeyAdapter){}

        public void addIdKeyListener(KeyAdapter idKeyAdapter){}

        protected boolean hasEmptyRequiredFields(){ }

c. Instrumentar una manera de obtener información de la vista basándose en alguna de las estrategias del segundo escenario, en el paso 1.b.

2. Crear el nuevo modelo, que sea subclase de *Model* o *MetadataModel*, según la finalidad:
    a. Si el modelo es subclase de *MetadataModel*, seguir el paso 3 del primer escenario, omitiendo el índice "c".
    b. Si el modelo no es para metadatos, definir la lógica aplicativa del modelo según convenga, no olvidando incluir constantes y métodos de actualización y notificación a vistas.

3. Crear el nuevo controlador, que sea subclase de *Controller* o *MetadataController*, según la finalidad:
    a. Si se extiende *MetadataController*, seguir el paso 4 del primer escenario.
    b. Si no es para manejo de metadatos, seguir sólo el paso 4.a del primer escenario.

4. Crear nuevo *manager*, que sea subclase de *DialogManager:*
    a. Si el *manager* es para manejo de metadatos, seguir primero el paso 5 del primer escenario.
    b. Instrumentar una manera de obtener la salida deseada del diálogo. Generalmente será necesario llamar métodos del modelo en este paso.
    c. Instrumentar alguna manera para desplegar datos iniciales a visualizarse en el diálogo.

Para usar estos componentes se puede optar por cualquiera de las dos siguientes opciones:

- usar el método *public int showDialog(String title)* si se desea mostrar el diálogo como parte del módulo central de VELOAT;
- usar el método public *int showDialog(Frame parent, String title, boolean modal)* si el diálogo se quiere usar de manera independiente.

A continuación se muestran ejemplos de uso independiente del módulo central de VELOAT para *VoiceRecordingManager* y *LomManager*:

```
VoiceRecordingManager m = new VoiceRecordingManager();
If(m.showDialog(null,"Grabación de voz",true)==BaseDialog.OK)
{
      File file = (File)m.getOutput();
      // do whatever you want with your voice recording
}
```

```
LomManager m = new LomManager(someContentToAnalize, aLomObject);
LomManager.setMetadataGenerationEnabled(true);
if(m.showDialog(null,"Edición de metadatos",true)==BaseDialog.OK)
{
      Lom lom = (Lom)m.getOutput();
      // do whatever you want with the Lom metadata
}
```