Referencia Nibiru

21 de octubre de 2011

http://code.google.com/p/nibiru/

Parte I

Introducción

1. Objetivo del framework

El objetivo es brindar un marco que facilite el desarrollo de aplicaciones modulares. Se establecen las siguientes metas para lograr dicho objetivo:

- Proveer una capa de abstracción de las diferentes tecnologías usadas, para evitar el acoplamiento.
- Brindar servicios que sean comunes a las aplicaciones de negocio, como ser ABMs, reportes, workflow, gestión de transacciones, seguridad o internacionalización.
- Proveer mecanismos de actualización dinámica para que el sistema se pueda actualizar en caliente.
- Implementar patrones que faciliten resolver problemas de una manera estructurada. Pero a la vez no forzar al usuario a implementar una solución dada.
- Posibilitar la comunicación desacoplada entre módulos.
- No reinventar la rueda. Crear capas de abstracción pero usar en lo posible tecnologías existentes.

2. Arquitectura

En esta sección se explican las decisiones de arquitectura tomadas.

2.1. Patrón IoC

A fin de desacoplar cada componente del contenedor y de otros componentes, las dependencias de cada componente son inyectadas (patrón IoC).

2.2. Patrón MVP

El modelo utilizado para la capa de presentación es el patrón MVP, en su variante de vista pasiva. Esto permite tener desacoplados los presenters entre si mediante un bus de eventos y a su vez tener desacoplada la implementación de la vista. Google también hace una buena descripción de este patrón.

Además se llevó la idea de abstraer la vista un paso más allá, creando abstracciones para los componentes más comunes. De esta manera, el usuario puede optar por crear una vista genérica o una vista utilizando las ventajas particulares de una tecnología dada.

2.3. División entre API e implementación

A fin de facilitar el desacoplamiento entre implementaciones de distintos módulos, se definieron dos tipos de módulos:

- API: Contienen interfaces de componentes a ser expuestos a otros componentes. Por convención de nombre, finalizan en ".api".
- Implementación: Contienen implementaciones de las APIs. Por convención de nombres tienen el mismo nombre del API que implementan pero cambiando el ".api" final por algo descriptivo de la implementación.

En general, cualquier módulo sólo puede acceder a otro a través de un API. La excepción a esta regla son los módulos con utilidades, que no exponen servicios en sí, sino que sólo exportan clases de uso general.

Por convención de nombres, las implementaciones de APIs que no dependan de una tecnología en particular tendrán el sufijo ".generic".

2.4. Puntos de extensión

El sistema tiene un mecanismo de puntos de extensión que permite agregar o quitar funcionalidad de manera dinámica. La idea se tomó de la plataforma Eclipse, pero intentando armar un mecanismo más simple.

2.5. Extensión mediante scripting

El framework permite la personalización mediante scripting. Esto, en combinación con el mecanismo de puntos de extensión, permite que el usuario agregue funcionalidad al sistema desde la administración del mismo, sin necesidad de compilar, empaquetar o instalar nada.

2.6. Plataforma Java

Se optó por Java debido a que actualmente es la plataforma de más amplia difusión dentro de las aplicaciones empresariales, además de ser fácilmente portable a distintos ambientes, disponer de innumerables frameworks y librerías, etc.

2.7. OSGi / Spring DM

Se optó por usar OSGi debido a que brinda un mecanismo para gestión dinámica de módulos. Se utilizó Spring DM porque brinda muchas facilidades para implementar el patrón IoC bajo OSGi. No se utilizó Gemini porque al momento de iniciar el proyecto el mismo estaba muy inmaduro aún.

Utilizando estas tecnologías, los componentes compartidos son expuestos mediante servicios OSGi. La división entre API e implementación permite además el cambio en caliente de servicios, al no acceder los componentes cliente a la clase concreta de la implementación. Por otro lado, Spring DM brinda proxies que hacen que dichos cambios en caliente sean transparentes para el código cliente.

De cualquier modo, casi todos los componentes son independientes de OSGi y de Spring, gracias al patrón IoC (salvo los que implementan funcionalidades específicas de Spring).

3. Primeros pasos

3.1. Software requerido

1. Java (http://www.java.com/es/download/).
2. Eclipse (http://www.eclipse.org/).
3. Maven (http://maven.apache.org/).
4. Algún cliente GIT (http://git-scm.com/). Nosotros usamos EGit.

3.2. Instalación

1. Clone el proyecto como se explica en http://code.google.com/p/nibiru/source/checkout.
2. Ejecutar "mvn eclipse:eclipse" desde el directorio para generar el proyecto Eclipse a partir de los archivos de Maven y descargar los JARs del target platform.
3. Vaya a ar.com.oxen.sample/ar.com.oxen.sample.targetplatform y ejecute "mvn compile" a fin de crear el target platform a partir de las dependencias Maven.
4. Importar los proyectos creados desde Eclipse.
5. En preferencias, activar el target platform de Nibiru. Seleccionar la opción "reload" para que tome en cuenta los JARs descargados.
6. Ejecutar el launch de aplicación OSGi que se llama "Nibiru Test". Eclipse agrega por defecto los proyectos de tipo plugin (OSGi) que estén en el workspace, de manera que aunque exista un JAR con el mismo proyecto, el proyecto fuente tiene precedencia.

Si no quiere descargar los fuentes completos, puede ejecutar desde los binarios descargando la aplicación de ejemplo precompilada: http://nibiru.googlecode.com/files/sampleapp.zip. Aunque la aplicación ya esté empaquetada, se debe crear el target platform y se debe ejecutar desde Eclipse.

3.3. Proyecto de ejemplo

Al ejecutar la aplicación se creará una base de datos H2 en un directorio nibiruDb en su home directory. Los usuarios de Windows deberían modificar el archivo ar.com.oxen.nibiru.sample/ar.com.oxen.nibiru.sample.datasource.fragment/src/main/resources/databa para especificar la ubicación de la base de datos.

La aplicación de ejemplo usa un servicio de autenticación de prueba. Ingrese con usuario "guest", clave "guest".

TODO: Simplificar el armado de un proyecto. Opciones:

1. *Hacer un namespace handler.*
2. *Armar anotaciones (aunque no se cómo encajaría esto con Spring DM).*
3. *Usar directamente Guice+Peaberry (http://code.google.com/p/peaberry/)*

Parte II

Módulos

4. Aplicación base

El bundle ar.com.oxen.nibiru.application.api contiene las interfaces utilizadas para implementar funciones básicas de la aplicación como ser login, ventana de "acerca de", etc.

La idea es que una implementación de este bundle provea la base para levantar la aplicación y toda la funcionalidad extra se agregue mediante otros módulos.

Este módulo contiene los factories para los presentadores:

```
package ar.com.oxen.nibiru.application.api;

import ar.com.oxen.nibiru.application.api.about.AboutView;
import ar.com.oxen.nibiru.application.api.login.LoginView;
import ar.com.oxen.nibiru.application.api.main.MainView;
import ar.com.oxen.nibiru.ui.api.mvp.Presenter;

/**
 * Presenter factory for common application functionality.
 */
public interface ApplicationPresenterFactory {
        /**
         * Builds the presenter for login window.
         *
         * @return The presenter
         */
        Presenter<LoginView> buildLoginPresenter();

        /**
         * Builds the presenter for main window.
         *
         * @return The presenter
         */
        Presenter<MainView> buildMainPresenter();

        /**
         * Builds the presenter for about window.
         *
         * @return The presenter
         */
        Presenter<AboutView> buildAboutPresenter();
```

```
}
```

Y para las vistas de la aplicación:

```
package ar.com.oxen.nibiru.application.api;

import ar.com.oxen.nibiru.application.api.about.AboutView;
import ar.com.oxen.nibiru.application.api.login.LoginView;
import ar.com.oxen.nibiru.application.api.main.MainView;

/**
 * View factory for common application functionality.
 */
public interface ApplicationViewFactory {
        /**
         * Builds the view for login window.
         *
         * @return The view
         */
        LoginView buildLoginView();

        /**
         * Builds the view for main window.
         *
         * @return The view
         */
        MainView buildMainView();

        /**
         * Builds the view for about window.
         *
         * @return The view
         */
        AboutView buildAboutView();
}
```

TODO: Los presentadores y las vistas deberían ir en módulos separados.

El bundle ar.com.oxen.nibiru.application.generic provee una implementación genérica de los componentes base de la aplicación.

5. Puntos de extensión

Las interfaces para puntos de extensión se encuentran en el bundle ar.com.oxen.nibiru.extensionpoint.api. El diseño es simple: cada punto de extension tiene una interfaz dada y un nom-

bre. Y además, las extensiones pueden activarse o desactivarse en tiempo de ejecución.

A fin de realizar una acción cada vez que una extensión se agregue o se remueva, se debe utilizar la interfaz ExtensionTracker:

```
package ar.com.oxen.nibiru.extensionpoint.api;

/**
 * Callback for tracking extension status.
 *
 * @param <T>
 *            The extension type
 */
public interface ExtensionTracker<T> {
        /**
         * Callback method called when a new extension is registered.
         *
         * @param extension
         *            The extension
         */
        void onRegister(T extension);

        /**
         * Callback method called when an existing extension is unregistered.
         *
         * @param extension
         *            The extension
         */
        void onUnregister(T extension);
}
```

que provee los callbacks necesarios para dichos eventos. Los ExtensionTrackers deben ser registrados en el servicio ExtensionPointManager:

```
package ar.com.oxen.nibiru.extensionpoint.api;

/**
 * Service for managing extensions.
 */
public interface ExtensionPointManager {
        /**
         * Registers an extension under a name and an interface
         *
         * @param <K>
         *            The extension point interface
         * @param extension
         *            The extension
```

```
     * @param extensionPointName
     *              The extension point name
     * @param extensionPointInterface
     *              The extension point interface
     */
    <K> void registerExtension(K extension, String extensionPointName,
                    Class<K> extensionPointInterface);

    /**
     * Un-registers an extension.
     *
     * @param extension
     *              The extension.
     */
    void unregisterExtension(Object extension);

    /**
     * Registers a tracker for a given extension type and name.
     *
     * @param <T>
     *              The type parametrized on the tracker
     * @param <K>
     *              The extension point interface
     * @param tracker
     *              The tracker
     * @param extensionPointName
     *              The extension point name
     * @param extensionPointInterface
     *              The extension point interface
     */
    <T, K extends T> void registerTracker(ExtensionTracker<T> tracker,
                    String extensionPointName, Class<K> extensionPointInterf
}
```

La interfaz ExtensionPointManager también provee métodos para registrar nuevas extensiones y dar de baja extensiones existentes.

El bundle ar.com.oxen.nibiru.extensionpoint.spring tiene una implementación basada en Spring DM y servicios OSGi de los puntos de extensión. Bajo esta implementación, cada punto de extensión simplemente se implementa mediante un servicio OSGi con una propiedad llamada "extensionPoint" utilizada para indicar el nombre del punto de extensión sobre el cual se agregará la funcionalidad.

6. Bus de eventos

Varios módulos hacen uso del bus de evento. El bus de eventos se accede utilizando la interfaz ar.com.oxen.commons.eventbus.api.EventBus, que no pertenece al proyecto Nibiru sino a Oxen Java Commons. En este proyecto también hay una implementación (bastante) simple de esa interfaz.

7. Módulos

Como se dijo antes, el framework está pensado para que la funcionalidad se añada a modo de módulos independientes.

El proyecto ar.com.oxen.nibiru.module.utils provee clases de utilidad para tal fin. Típicamente cada módulo tendrá un componente encargado de configurar dicho módulo al arranque. Para tal fin, este proyecto provee la clase AbstractModuleConfigurator de la cual se puede heredar para crear dichos configuradores.

```
package ar.com.oxen.nibiru.module.utils;

import java.util.Collection;
import java.util.LinkedList;

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.nibiru.extensionpoint.api.ExtensionPointManager;
import ar.com.oxen.nibiru.ui.api.mvp.Presenter;
import ar.com.oxen.nibiru.ui.api.mvp.View;

/**
 * Base class for module configurators.
 *
 * @param <VF>
 *            The view factory class
 * @param <PF>
 *            The presenter factory class
 */
public abstract class AbstractModuleConfigurator<VF, PF> {
        private ExtensionPointManager extensionPointManager;
        private Collection<Object> registeredExtensions = new LinkedList<Object>
        private EventBus eventBus;
        private VF viewFactory;
        private PF presenterFactory;

        /**
         * Starts the module. This method must be externally called (for example
         * with init-method attribute on Spring context XML).
```

```
 */
public void startup() {
        this.eventBus.subscribeAnnotatedObject(this);
        this.configure();
}

/**
 * Same as startup, but for shutdown.
 */
public void shutdown() {
        /* Remove all the extensions */
        for (Object extension : this.registeredExtensions) {
                this.extensionPointManager.unregisterExtension(extension
        }
        this.registeredExtensions.clear();

        this.unconfigure();
}

/**
 * Abstract method to be override in order to customize module
 * configuration.
 */
protected void configure() {
}

/**
 * Abstract method to be override in order to customize module
 * un-configuration.
 */
protected void unconfigure() {
}

/**
 * Activates a view/presenter. Typically this method will be called from
 * subclasses upon the receiving of an event from the bus in order to
 * navigate to a given window.
 *
 * @param <V>
 *            The view type
 * @param view
 *            The view
 * @param presenter
 *            The presenter
 */
protected <V extends View> void activate(V view, Presenter<V> presenter)
```

```
        presenter.setView(view);
        presenter.go();
        view.show();
}

/**
 * Registers an extension under a name and an interface. The extension w
 * be automatically un-published when the module will be unloaded.
 *
 * @param <K>
 *            The extension point interface
 * @param extension
 *            The extension
 * @param extensionPointName
 *            The extension point name
 * @param extensionPointInterface
 *            The extension point interface
 */
protected <K> void registerExtension(K extension,
                String extensionPointName, Class<K> extensionPointInterf
        this.extensionPointManager.registerExtension(extension,
                        extensionPointName, extensionPointInterface);
        this.registeredExtensions.add(extension);
}

public void setEventBus(EventBus eventBus) {
        this.eventBus = eventBus;
}

protected EventBus getEventBus() {
        return eventBus;
}

protected VF getViewFactory() {
        return viewFactory;
}

public void setViewFactory(VF viewFactory) {
        this.viewFactory = viewFactory;
}

protected PF getPresenterFactory() {
        return presenterFactory;
}

public void setPresenterFactory(PF presenterFactory) {
```

```
                this.presenterFactory = presenterFactory;
        }

        public void setExtensionPointManager(
                        ExtensionPointManager extensionPointManager) {
                this.extensionPointManager = extensionPointManager;
        }
}
```

Se debe inyectar las dependencias necesarias y disparar el método startup() en el arranque. Al detener el módulo se debe disparar el método shutdown(). Los métodos configure() y unconfigure() pueden ser implementados a fin de proveer lógica personalizada de configuración en el arranque y en la detención, respectivamente.

Típicamente este componente configurará la navegación entre distintas pantallas del módulo. Para esto, la clase AbstractModuleConfigurator provee acceso al bus de eventos (que debe ser inyectado) y se pone a si mismo como listener de dicho bus. De manera que pueden agregarse métodos de manejo de eventos anotados con @EventHandler. Para mostrar una vista/presentador se puede usar el método activate().

Además la clase provee métodos para registrar extensiones (debe estar inyectado el ExtensionPointManager). Dichas extensiones son removidas automáticamente cuando el módulo es dado de baja.

En cuanto a los menúes, son implementados mediante puntos de extensión. De modo que solamente es necesario registrar extensiones con las siguiente interfaz:

```
package ar.com.oxen.nibiru.ui.api.extension;

/**
 * Extension that represents an item on the menu.
 */
public interface MenuItemExtension {
        /**
         * @return The item name
         */
        String getName();

        /**
         * @return The position (lower numbers are shown first)
         */
        int getPosition();

        /**
         * Method to be executed when the menu is created.
         */
```

```
        void onClick();
}
```

o bien con:

```
package ar.com.oxen.nibiru.ui.api.extension;

/**
 * Extension that represents a menu that can contain other menus.
 */
public interface SubMenuExtension {
        /**
         * @return The sub-menu name
         */
        String getName();

        /**
         * @return The position (lower numbers are shown first)
         */
        int getPosition();

        /**
         * @return The extension point name where entries of this sub-menu shoul
         *          added.
         */
        String getExtensionPoint();
}
```

Se debe definir un nombre de punto de extensión para cada menú. El punto de extensión para el menú principal es ar.com.oxen.nibiru.menu.

Vale la pena notar que en el bundle ar.com.oxen.nibiru.ui.utils hay implementaciones simples de estas interfaces.

8. Sesión

Generalmente las aplicaciones tienen algún tipo de información de sesión. Esto es, datos que son propios del usuario que esté conectado en un momento dado. Típicamente, en una aplicación Web, esta información se almacena en la sesión HTTP.

A fin de apoyar la meta de mantener los distintos componentes desacoplados de la implementación, el proyecto ar.com.oxen.nibiru.session.api provee una interfaz genérica para una sesión.

```
package ar.com.oxen.nibiru.session.api;
```

```
/**
 * Component holding session data.
 */
public interface Session {
        /**
         * Gets an object from session data.
         *
         * @param <T>
         *              The object type
         * @param key
         *              The object key (must be unique)
         * @return The object
         */
        <T> T get(String key);

        /**
         * Puts an object into session data.
         *
         * @param key
         *              The object key (must be unique)
         * @param value
         *              The object
         */
        void put(String key, Object value);

        /**
         * Removes an object from session data.
         *
         * @param key
         *              The object key (must be unique)
         */
        void remove(String key);

        /**
         * @return An String identifying the session.
         */
        String getId();

        /**
         * @return A mutex that can be used in order to synchronize concurrent
         *          (threaded) session access
         */
        Object getMutex();

        /**
```

```
         * Registers a listener for session destruction.
         *
         * @param name
         *             The callback name (must be unique)
         * @param callback
         *             The callback
         */
        void registerDestructionCallback(String name, Runnable callback);

        /**
         * @return True if the session is valid
         */
        boolean isValid();
}
```

El proyecto ar.com.oxen.nibiru.session.spring.http provee acceso a la sesión HTTP utilizando componentes de Spring (filtros de Servlet que brindan acceso a la sesión a través de un ThreadLocal).

El proyecto ar.com.oxen.nibiru.session.spring.scope provee un scope de Spring que permite declarar beans en el contexto de Spring que sean almacenados en la sesión provista por nibiru. En conjunto con el tag <aop:scoped-proxy/> provisto por Spring, este mecanismo permite qeu beans que son almacenados en la sesión sean inyectados de manera transparente a beans que son singleton.

Por ejemplo:

```
...

<osgi:reference id="nibiruSession"
    interface="ar.com.oxen.nibiru.session.api.Session" />

<bean class="org.springframework.beans.factory.config.CustomScopeConfigurer">
  <property name="scopes">
    <map>
      <entry key="nibiruSession">
        <bean
            class="ar.com.oxen.nibiru.session.spring.scope.SessionScope">
          <property name="session" ref="nibiruSession" />
        </bean>
      </entry>
    </map>
  </property>
</bean>

<bean name="vaadinApplication" scope="nibiruSession"
    class="ar.com.oxen.nibiru.ui.vaadin.application.NibiruApplication">
  <property name="eventBus" ref="eventBus" />
```

```
  <property name="localeHolder" ref="localeHolder" />
  <aop:scoped-proxy />
</bean>

...
```

9. Conversaciones

Un escenario común en las aplicaciones de negocios es que los usuarios operen sobre un conjunto de datos durante un intervalo de tiempo dado y finalmente confirmen las operaciones pendientes sobre ellos o cancelen todo el proceso. La conversación (proyecto ar.com.oxen.nibiru.conversation.api) sirve como abstracción de este concepto:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Interface representing a conversation between the user and the application.
 */
public interface Conversation {
        /**
         * Finishes the conversation OK. Typically, this action is called when a
         * user clicks an "accept" button in order to confirm database changes,
         */
        void end();

        /**
         * Cancels the conversation. Typically called when the user presses a
         * "cancel" button.
         */
        void cancel();

        /**
         * Registers a conversation status tracker.
         *
         * @param tracker
         *            The tracker
         */
        void registerTracker(ConversationTracker tracker);

        /**
         * Activates the conversation and executes the code provided by the
         * callback. Code called from the callback can access the conversation u
         * the {@link ConversationAccessor} service.
```

```
     *
     * @param <T>
     *            The type to be returned by the callback
     * @param callback
     *            The callback
     * @return The object returned by the callback
     */
    <T> T execute(ConversationCallback<T> callback);

    /**
     * Gets an object from conversation data.
     *
     * @param <T>
     *            The object type
     * @param key
     *            The object key (must be unique)
     * @return The object
     */
    <T> T get(String key);

    /**
     * Puts an object into conversation data.
     *
     * @param key
     *            The object key (must be unique)
     * @param value
     *            The object
     */
    void put(String key, Object value);

    /**
     * Removes an object from conversation data.
     *
     * @param key
     *            The object key (must be unique)
     */
    void remove(String key);
}
```

La conversación provee una forma de desacoplar la interfaz de usuario de la implementación de los distintos servicios que requieran de información de conversación. Por ejemplo, supongamos que estamos usando el módulo de ABM con la implementación JPA del servicio. La capa de interfaz de usuario crea una conversación al abrir el presentador. Ante cada llamada al servicio, la implementación del mismo extrae de la conversación el EntityManager activo. De esta manera, las capas superiores no necesitan saber los detalles sobre la infor-

mación de conversación que necesitan las capas inferiores.

Para implementar este proceso, el cliente (usualmente la capa de presentación) crea una conversación utilizando el factory:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Conversation factory.
 */
public interface ConversationFactory {
        /**
         * Builds a new conversation.
         *
         * @return The conversation
         */
        Conversation buildConversation();
}
```

y cada vez que accede a un servicio que requiera de información de conversación, lo hace mediante el método execute(), que recibe un callback con un método doInConversation(), que ejecutará luego de activar la conversación:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Conversation callback. Used to run code that can access the active
 * conversation using {@link ConversationAccessor}.
 *
 * @param <T>
 */
public interface ConversationCallback<T> {
        /**
         * Method to be executed when conversation is activated.
         *
         * @param conversation
         *            The active conversation
         * @return Anything that the callback would want to return
         * @throws Exception
         *             At any error
         */
        T doInConversation(Conversation conversation) throws Exception;
}
```

Finalmente, el cliente puede invocar el método end() o el método cancel(), según desee finalizar o cancelar la conversación.

Del lado de las capas inferiores, es posible acceder a la conversación activa mediante el servicio ConversationAccessor:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Service used to access current active conversation.
 */
public interface ConversationAccessor {
        /**
         * @return The current conversation
         */
        Conversation getCurrentConversation();
}
```

Mediante los métodos put() y get(), el componente puede escribir y leer valores en la conversación. En caso de que se desee realizar una acción al finalizar o cancelar una conversación, se puede utilizar el método registerTracker() para registrar un callback:

```
package ar.com.oxen.nibiru.conversation.api;

/**
 * Listener for tracking conversation life cycle.
 *
 */
public interface ConversationTracker {
        /**
         * Called when conversation finishes OK.
         *
         * @param conversation
         *              The finished conversation
         */
        void onEnd(Conversation conversation);

        /**
         * Called when conversation is canceled.
         *
         * @param conversation
         *              The canceled conversation
         */
        void onCancel(Conversation conversation);
}
```

La idea de establecer un mecanismo de conversaciones proviene de Seam, pero se realizaron algunas modificaciones. En primer lugar, se buscó hacer el diseño más simple y que no esté orientado específicamente a aplicaciones Web. Por ejemplo, las conversaciones de Seam son jerárquicas, mientras que las de Nibiru no lo son. Incluso se pensó en unificar el concepto de conversación con el de sesión y hacerlo jerárquico (siendo la sesión la conversación principal), pero

esto añadiría complejidad a la semántica de las conversaciones y forzaría una unificación poco elegante de interfaces, sin aportar beneficios.

El módulo ar.com.oxen.nibiru.conversation.Overfull genérica de los servicios de conversación.

TODO: Las conversaciones pueden hacer que sea practicamente imposible serializar la sesión.

10. Persistencia

Para persistencia se utiliza JPA. Si bien existen mútliples mecanismo de persistencia en la plataforma Java, JPA es el más difundido. Por este motivo se eligió esta especificación por sobre otros mecanismos. De todas maneras, nada impide que se implementen otros servicios de persistencia utilizando alguna tecnología diferente (claro que esto impicaría implementar nuevamente los módulos que dependan de JPA).

Al ser JPA en sí mismo un API, no se definió un API propio de Nibiru. En cambio, se expone como servicio una instancia de javax.persistence.EntityManagerFactory, de la especificación JPA. El bundle ar.com.oxen.nibiru.jpa.spring provee 2 implementaciones de dicho servicio que utilizan clases de Spring:

1. ConversationEntityManagerFactory: Obtiene el EntityManager de la conversación activa y si no existe, lo crea. Actualmente este componente es expuesto como servicio.
2. SessionEntityManagerFactory: Obtiene el EntityManager de la sesión y si no existe, lo crea. Actualmente se está evaluando si este componente debe eliminarse (fue la implementación original del servicio).

Debido a que JPA requiere que se especifique en un archivo META-INF/persistence.xml las clases a persistir, deben crearse fragmentos OSGi para agregar dicho archivo al bundle del servicio JPA. Esto tiene como inconveniente que en dicho archivo se deben incluir las clases a persistir por los diferentes módulos. Ver proyecto ar.com.oxen.nibiru.sample.jpa.fragment para tomar como ejemplo.

En cuanto al acceso a base de datos, se expone un servicio con interfaz javax.sql.DataSource. En este caso tampoco fue necesario definir un API específico de Nibiru. El bundle ar.com.oxen.nibiru.datasource.dbcp provee una implementación con DBCP. La configuración de conexión a la base de datos, así como la visibilidad del driver JDBC, se agregan también mediante fragmentos OSGi. Ver proyecto ar.com.oxen.nibiru.sample.datasource.fragment.

11. Interfaz de usuario

El bundle ar.com.oxen.nibiru.ui.api contiene las interfaces para capa de presentación. El esquema apunta a que la vista se construya utilizando el patrón MVP (vista pasiva). Dentro del paquete principal tenemos 3 sub-paquetes:

1. extension: Contiene interfaces a implementar por las extensiones de UI (actualmente menú y sub-menú - ver sección Módulos para más detalles).
2. mvp: Contiene las interfaces a utilizar para implementar el patrón MVP: Presenter, View y todas las necesarias para acceder a datos y a eventos (HasValue, HasClickHandler, ClickHandler, etc.).
3. view: Contiene interfaces para abstracción de componentes de vista. Estas interfaces se usan cada vez que se quiere acceder de forma genérica a un widget específico. Por ejemplo, un botón o un campo de texto. La idea es que haya adaptadores para los widgets de las diferentes tecnologías de UI.

Bajo este esquema, el usuario tiene dos opciones para crear una vista:

1. De manera genérica, es decir, utilizando una implementación de ar.com.oxen.nibiru.ui.api.view.ViewFactor para acceder a interfaces genéricas de los widgets. De esta manera se puede constuir una interfaz limitada, pero se puede cambiar fácilmente la tecnología subyacente.
2. Utilizando una tecnología específica y hacer que implemente la interfaz de la vista. De esta manera se pueden aprovechar características propias de la tecnología y utilizar editores gráficos. En contraste, el cambio de tecnología implicaría mas trabajo.

Como el modelo MVP propuesto es de vista pasiva, el presentador simplemente tiene una referencia a una interfaz que representa a la vista (en el caso de Google usan el término Display). Esto permite usar indistintamente cualquiera de los dos enfoques, sin cambiar el presentador.

En síntesis, las interfaces principales del MVP son Presenter:

```
package ar.com.oxen.nibiru.ui.api.mvp;

/**
 * A presenter. The presenter should contain the presentation logic, in order to
 * keep it decoupled from the view.
 *
 * @param <V>
 *            The view type.
 */
public interface Presenter<V extends View> {
```

```
        /**
         * Activates the presenter. This method is called after setting the view
         * Typically, this method will add listeners for presentation logic that
         * reacts to view events.
         */
        void go();

        /**
         * @param view
         *            The view to be used with the presenter
         */
        void setView(V view);
}
```

y View:

```
package ar.com.oxen.nibiru.ui.api.mvp;

/**
 * A view. Implementations of this interface shouldn't contain presentation
 * logic. Instead, display-related logic, such as layout setup, text
 * internationalization, etc should be responsibility of View implementations.
 *
 */
public interface View {
        /**
         * Shows the view.
         */
        void show();

        /**
         * Closes the view.
         */
        void close();
}
```

En el método go() de Presenter se debe incluir la lógica de capa de presentación.

Las interfaces de abstracción de widgets (paquete ar.com.oxen.nibiru.ui.api.view) son variadas. Pero todas deberían instanciarse por medio de una implementación de ViewFactory:

```
package ar.com.oxen.nibiru.ui.api.view;

/**
 * Builds components (widgets, windows, etc) to be used in views. The purpose of
 * this interface is hiding UI framework specific implementations.
 */
```

```
public interface ViewFactory {
        /**
         * Builds a main window.
         *
         * @return The main window.
         */
        MainWindow buildMainWindow();

        /**
         * Builds a window.
         *
         * @return The window
         */
        Window buildWindow();

        /**
         * Builds a label.
         *
         * @param <T>
         *            The type of data to be shown by the label. Typically Strin
         * @param type
         *            The class of data to be shown by the label. Typically Stri
         * @return The label
         */
        <T> Label<T> buildLabel(Class<T> type);

        /**
         * Builds a button.
         *
         * @return The button.
         */
        Button buildButton();

        /**
         * Builds a text field.
         *
         * @param <T>
         *            The type of data to be shown by the text field. Typically
         *            String.
         * @param type
         *            The class of data to be shown by the text field. Typically
         *            String.
         * @return The text field
         */
        <T> TextField<T> buildTextField(Class<T> type);
```

```
/**
 * Builds a password field.
 *
 * @param <T>
 *            The type of data to be shown by the password field. Typica
 *            String.
 * @param type
 *            The class of data to be shown by the password field. Typic
 *            String.
 * @return The password field
 */
<T> PasswordField<T> buildPasswordField(Class<T> type);

/**
 * Builds a multiline text area.
 *
 * @param <T>
 *            The type of data to be shown by the password field. Typica
 *            String.
 * @param type
 *            The class of data to be shown by the password field. Typic
 *            String.
 * @return The text area
 */
<T> TextArea<T> buildTextArea(Class<T> type);

/**
 * Builds a date field.
 *
 * @return The date field
 */
DateField buildDateField();

/**
 * Builds a time field.
 *
 * @return The time field
 */
TimeField buildTimeField();

/**
 * Builds a check box.
 *
 * @return The check box
 */
CheckBox buildCheckBox();
```

```
/**
 * Builds a combo box.
 *
 * @param <T>
 *            The type of data to be shown by the combo.
 * @param type
 *            The class of data to be shown by the combo.
 * @return The combo box
 */
<T> ComboBox<T> buildComboBox(Class<T> type);

/**
 * Builds a list select.
 *
 * @param <T>
 *            The type of data to be shown by the list select.
 * @param type
 *            The class of data to be shown by the list select.
 * @return The list select
 */
<T> ListSelect<T> buildListSelect(Class<T> type);

/**
 * Builds a table.
 *
 * @return The table
 */
Table buildTable();

/**
 * Builds a panel with vertical layout.
 *
 * @return The panel.
 */
Panel buildVerticalPanel();

/**
 * Builds a panel with horizontal layout.
 *
 * @return The panel.
 */
Panel buildHorizontalPanel();

/**
 * Builds a panel with form layout.
```

```
         *
         * @return The panel.
         */
        Panel buildFormPanel();

        /**
         * Builds a tabbed panel.
         *
         * @return The panel
         */
        Panel buildTabPanel();
}
```

El proyecto ar.com.oxen.nibiru.ui.vaadin contiene adaptadores y su correspondiente factory para implementar las interfaces de ar.com.oxen.nibiru.ui.api.view utilizando Vaadin.

El proyecto ar.com.oxen.nibiru.ui.utils contiene clases genéricas para uso en la interfaz de usuario. En su mayoría, contiene clases abstractas para heredar y crear presentadores, vistas, extensiones, etc. Pero también decoradores y clases de uso genérico.

12. Seguridad

Las interfaces para acceder a los servicios de seguridad (autenticación y autorización) se encuentran en el proyecto ar.com.oxen.nibiru.security.api. Actualmente se soporta autenticación por usuario/clave y autorización por roles.

La autenticación se realiza por medio de la interfaz AuthenticationService:

```
package ar.com.oxen.nibiru.security.api;

/**
 * Service for authenticating users.
 */
public interface AuthenticationService {
        /**
         * Performs an user log-on.
         *
         * @param user
         *             The user name
         * @param password
         *             The password
         * @throws BadCredentialsException
         *              If the user name and/or the password is not valid
         */
```

```
        void login(String user, String password) throws BadCredentialsException;

        /**
         * Performs an user log-off.
         */
        void logout();

        /**
         * @return The login name of the logged user (if any).
         */
        String getLoggedUserName();
}
```

Mientras que la autorización se lleva a cabo mediante AuthorizationService:

```
package ar.com.oxen.nibiru.security.api;

/**
 * Service for authorizing actions and users.
 *
 */
public interface AuthorizationService {
        /**
         * Checks if the logged user has a given role.
         *
         * @param role
         *             The role name.
         * @return True if the user has the role
         */
        boolean isCallerInRole(String role);
}
```

TODO: Habría que pensar bien el esquema de autorización. Si los módulos se van a desarrollar de forma independiente, cómo se evita la colisión de roles? Depende de la implementación asociar los roles específicos de cada módulo a un rol general? (como en EJB) O simplemente que cada modulo le ponga un prefijo al rol? (como se está haciendo con la internacionalizacion).

Hasta el momento no se ha hecho ninguna implementación de seguridad, sólo hay un bundle que da un servicio dummy (ar.com.oxen.nibiru.security.dummy).

13. Gestión de transacciones

Dado que existen mecanismos no intrusivos (mediante AOP), no se definió un API específico para este caso. Se podría llegar a definir en caso de que se opte por brindar una gestión programática de transacciones.

El bundle ar.com.oxen.nibiru.transaction.spring expone un TransactionManager de Spring como servicio OSGi. Dentro de cada bundle se pueden utilizar los mecanismos de AOP de Spring para, declarativamente, establecer las transacciones (inyectando el servicio TransactionManager).

Por ejemplo:

```
...

<osgi:reference id="transactionManager"
    interface="org.springframework.transaction.PlatformTransactionManager" />

<tx:advice id="txAdvice" transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="*" propagation="REQUIRED"/>
  </tx:attributes>
</tx:advice>

<bean  class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostP

<bean class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreato
  <property name="beanNames" value="dynamicBundleCrudManager"/>
  <property name="interceptorNames" value="txAdvice"/>
</bean>

...
```

Actualmente no se soportan todos los tags XML de Spring para gestión de transacciones, por lo que debe usarse un componente de tipo BeanNameAutoProxyCreator, como muestra el ejemplo.

TODO: El bundle expone un JpaTransactionManager. El nombre del proyecto debería decir "jpa" en algún lugar.

14. Internacionalizacion

En el proyecto ar.com.oxen.nibiru.i18n.api se encuentran las interfaces para internacionalización. Hay 3 servicios principales:

1. LocaleHolder: Utilizado para leer o escribir el Locale del usuario.
2. MessageSource: Utilizado para consultar mensajes por clave (con parámetros).
3. MessageProvider: Utilizado para proveer mensajes consultando por clave y Locale. Se realizó esta división para que cada módulo provea su MessageProvider. Típicamente habrá una implementación de MessageSource que los consolide.

Las 3 interfaces son muy simples, como se puede ver.

LocaleHolder:

```
package ar.com.oxen.nibiru.i18n.api;

import java.util.Locale;

/**
 * Service used to access the user locale.
 *
 */
public interface LocaleHolder {
        /**
         * Gets the user locale.
         *
         * @return The locale
         */
        Locale getLocale();

        /**
         * Sets the user locale.
         *
         * @param newLocale
         *             The locale
         */
        void setLocale(Locale newLocale);
}
```

MessageSource:

```
package ar.com.oxen.nibiru.i18n.api;

import java.util.Locale;

/**
 * Service for accessing i18n messages. Typically a view from a module will
 * access this service. Internally, implementation of this module should access
 * the current user locale with {@link LocaleHolder} and delegate on N
 * {@link MessageProvider}s in order to look for the searched message.
 */
public interface MessageSource {
        /**
         * Gets a i18n message
         *
         * @param code
         *             The message code
         * @param args
```

```
         *            The message arguments
         * @return The translated an parsed message. If the message is not found
         *         returns null.
         */
        String getMessage(String code, Object... args);

        /**
         * Returns a 18n message
         *
         * @param code
         *            The message code
         * @param locale
         *            The locale
         * @param args
         *            The message arguments
         * @return The translated an parsed message. If the message is not found
         *         null is returned.
         */
        String getMessage(String code, Locale locale, Object... args);
}
```

MessageProvider:

```
package ar.com.oxen.nibiru.i18n.api;

import java.util.Locale;

/**
 * A message provider. This interface is provided in order to allow i18n
 * modularity. Each module could provide its own MessageProvider. All the
 * MessageProviders would be consolidated by a single, generic
 * {@link MessageSource}.
 */
public interface MessageProvider {
        /**
         * Returns a 18n message
         *
         * @param code
         *            The message code
         * @param locale
         *            The locale
         * @param args
         *            The message arguments
         * @return The translated an parsed message. If the message is not found
         *         null is returned.
         */
        String getMessage(String code, Locale locale, Object... args);
```

}

El proyecto ar.com.oxen.nibiru.i18n.generic contiene una implementación genérica de MessageSource al cual se le inyecta el LocaleHolder y una lista de MessageProvider. Mediante Spring DM se puede inyectar una lista de servicios de tipo MessageProvider que se actualice dinámicamente ante la disponibilidad de nuevas instancias de dichos servicios. Este proyecto también contiene una implementación basada en ResoruceBundle de MessageProvider.

El proyecto ar.com.oxen.nibiru.i18n.session tiene una implementación de LocaleHolder que almacena el locale en la sesión de Nibiru.

15. ABM

El módulo de ABM (Alta, Baja y Modificación) apunta a facilitar la generación de pantallas de este tipo.

La funcionalidad de este módulo está distribuida entre varios bundles. La misma puede agruparse en 2 capas.

TODO: pensar cómo integrar seguridad al generador de ABMS.

15.1. Servicios de persistencia

Las interfaces necesarias para exponer servicios de persistencia se encuentran en el proyecto ar.com.oxen.nibiru.crud.manager.api.

La interfaz principal es CrudManager, que provee los métodos necesarios para generar dinámicamente una pantalla de ABM. En otras palabras, la idea es que haya un CrudManager por cada entidad sobre la cual se quiera realizar un ABM.

```
package ar.com.oxen.nibiru.crud.manager.api;

import java.util.List;

/**
 * Service for managing CRUD over entities.
 *
 * @param <T>
 *            The crud entity type.
 */
public interface CrudManager<T> {
        /**
         * Returns the entity type name.
         *
         * The name identifies the kind of entity being handled. This is useful,
```

```
     * example, in order to determine if a given entity is compatible with a
     * crud manager.
     *
     * @return The type name.
     */
    String getEntityTypeName();

    /**
     * Gets the fields to be shown in the entity list.
     *
     * @return A list with the fields
     */
    List<CrudField> getListFields();

    /**
     * Gets the fields to be shown in the entity form.
     *
     * @return A list with the fields
     */
    List<CrudField> getFormFields();

    /**
     * Reads all the entities.
     *
     * @return A list with the entities
     */
    List<CrudEntity<T>> findAll();

    /**
     * Reads entities filtering by a given field. Useful for parent-child
     * relations
     *
     * @return A list with the entities
     */
    List<CrudEntity<T>> findByfield(String field, Object value);

}
```

El módulo de ABM está pensado para que las pantallas de ABM puedan crearse sobre diversos tipos de entidades. A diferencia de un generador de ABMs típico, donde se generan pantallas para administrar tablas de una base de datos o sobre beans, en Nibiru se agrega un nivel de indirección. Esto permite que se creen implementaciones del servicio de persistencia provea acceso a beans JPA, a instancias de procesos de negocio, etc.

Las interfaces utilizadas para lograr este nivel de abstracción son CrudEntity

(que representa una entidad que está siendo editada) y CrudField (que representa un campo de dicha entidad).

```
package ar.com.oxen.nibiru.crud.manager.api;

/**
 * Represents an entity instance. This interface is used in order to hide entity
 * implementation. This way, CRUD engine could work over Java beans, BPM
 * processes, etc.
 *
 * @param <T>
 */
public interface CrudEntity<T> {
        /**
         * Reads a field value.
         *
         * @param field
         *              The field
         * @return The value
         */
        Object getValue(CrudField field);

        /**
         * Reads a field value.
         *
         * @param fieldName
         *              The field name
         * @return The value
         */
        Object getValue(String fieldName);

        /**
         * Writes a field value
         *
         * @param field
         *              The field
         * @param value
         *              The value
         */
        void setValue(CrudField field, Object value);

        /**
         * Writes a field value
         *
         * @param fieldName
         *              The field name
```

```
     * @param value
     *            The value
     */
    void setValue(String fieldName, Object value);

    /**
     * Gets the wrapped object.
     *
     * @return The entity object
     */
    T getEntity();

    /**
     * Returns the entity type name.
     *
     * The name identifies the kind of entity being handled. This is useful,
     * example, in order to determine if a given entity is compatible with a
     * crud manager.
     *
     * @return The type name.
     */
    String getEntityTypeName();

    /**
     * Returns the available values for a given field (for example, for usin
     * a combo box or a list select)
     *
     * @param field
     *            The field
     * @return An iterable for the values
     */
    Iterable<Object> getAvailableValues(CrudField field);

    /**
     * Returns the available values for a given field (for example, for usin
     * a combo box or a list select)
     *
     * @param fieldName
     *            The field name
     * @return An iterable for the values
     */
    Iterable<Object> getAvailableValues(String fieldName);
}

package ar.com.oxen.nibiru.crud.manager.api;
```

```
/**
 * Represents a field on a {@link CrudEntity}.
 *
 */
public interface CrudField {
        /**
         * @return The field name
         */
        String getName();

        /**
         * @return The field class
         */
        Class<?> getType();

        /**
         * @return Information for showing the field in a list.
         */
        ListInfo getListInfo();

        /**
         * @return Information for showing the field in a form.
         */
        FormInfo getFormInfo();

        /**
         * Information for showing the field in a list.
         */
        interface ListInfo {
                /**
                 * Determines a fixed width for the field column.
                 *
                 * @return The column width
                 */
                int getColumnWidth();

        }

        /**
         * Information for showing the field in a form.
         */
        interface FormInfo {
                /**
                 * Determines how the field should be represented (for example,
                 * form).
                 *
```

```
                * @return An element of widget type enumeration
                */
               WidgetType getWidgetType();

               /**
                * @return True if the field can't be modified
                */
               boolean isReadonly();

               /**
                * Determines how many characters can be set on the field. Appli
                * to widgets which holds Strings.
                *
                * @return The maximum length
                */
               int getMaxLength();

               /**
                * Returns the tab name where the widget must be shown.
                *
                * @return The tab name
                */
               String getTab();
       }
}
```

WidgetType enumera las formas en las que se puede mostrar un campo:

```
package ar.com.oxen.nibiru.crud.manager.api;

public enum WidgetType {
       TEXT_FIELD,
       PASSWORD_FIELD,
       TEXT_AREA,
       DATE_FIELD,
       TIME_FIELD,
       CHECK_BOX,
       COMBO_BOX,
       MULTISELECT
}
```

La abstracción no estaría completa si las acciones a realizar sobre las entidades no fueran configurables. Para este fin existe la interfaz CrudAction.

```
package ar.com.oxen.nibiru.crud.manager.api;

/**
```

```
 * Represents an action that can be applied on a CRUD. Abstracting the actions
 * allows the CRUD implementations to provide extra actions. This way, actions
 * are not limited to create, read, update an delete (so the module shouldn't be
 * called CRUD!!!), but can add action such as approve, reject, start, stop,
 * etc. In some cases, the action can require no entity (for example, "new"). In
 * other cases, it would be mandatory applying the action over an specific
 * {@link CrudEntity} ("edit", for example).
 *
 */
public interface CrudAction {
        String NEW = "new";
        String DELETE = "delete";
        String EDIT = "edit";
        String UPDATE = "update";

        /**
         * Gets the action name.
         *
         * @return The name
         */
        String getName();

        /**
         * Indicates if the action must be performed over an {@link CrudEntity}.
         *
         * @return True if a {@link CrudEntity} is required
         */
        boolean isEntityRequired();

        /**
         * Indicates if a user confirmation must be presented before performing
         * action.
         *
         * @return True if confirmation must be presented
         */
        boolean isConfirmationRequired();

        /**
         * Indicates if the action must be shown in list window.
         *
         * @return True if it must be shown
         */
        boolean isVisibleInList();

        /**
         * Indicates if the action must be shown in form window.
```

```
         *
         * @return True if it must be shown
         */
        boolean isVisibleInForm();
}
```

De esta manera las acciones no se limitan a alta, baja y modificaciones; sino que son extensibles. Un motor de workflow podría, por ejemplo, exponer acciones como "aprobar" o "rechazar".

Para que el esquema de ABM sea modular, las acciones a realizar sobre una entidad no son provistas directamente por el CrudManager sino que se usa el mecanismo de puntos de extensión. La interfaz CrudActionExtension permite que se implementen distintas extensiones que agregan posibles acciones a realizar sobre una entidad.

```
package ar.com.oxen.nibiru.crud.manager.api;

import java.util.List;

/**
 * Extension used to add actions to CRUD.
 *
 * @param <T>
 *            The {@link CrudEntity} type
 */
public interface CrudActionExtension<T> {
        /**
         * Get actions provided by this extension.
         *
         * @return A list with the actions
         */
        List<CrudAction> getActions();

        /**
         * Performs an action over a given entity. The action can create/update
         * entity. In that case, such entity is returned, otherwise it returns n
         * When a created/updated entity is returned, the CRUD should open a form
         * order to edit it. This can be useful, for example, for BPM
         * implementations that jumps from an activity to another.
         *
         * @param action
         *            The action
         * @param entity
         *            The entity (it can be null if the action doesn't require a
         *            entity)
         * @return The created/updated entity
```

```
         */
        CrudEntity<T> performAction(CrudAction action, CrudEntity<T> entity);
}
```

El bundle ar.com.oxen.nibiru.crud.manager.jpa contiene implementaciones que se basan en JPA. Se apoya en clases de ar.com.oxen.nibiru.crud.bean y de ar.com.oxen.nibiru.crud.utils. En lo posible usa reflection e información de JPA para retornar la información necesaria para el ABM, pero de no ser posible, se basa en las anotaciones de ar.com.oxen.nibiru.crud.bean.

15.1.1. Eventos

El API de ABMs provee eventos de uso común. Su propósito es que sean usados en la comunicación de los distintos componentes de ABM a través del bus de eventos.

El evento ManageCrudEntitiesEvent puede utilizarse para notificar que se desea administrar entidades de un tipo dado. Típicamente se dispara desde un menú.

```
package ar.com.oxen.nibiru.crud.manager.api;

/**
 * This is a generic event class for triggering entities management. The topic
 * should be used in order to identify the entity to be managed.
 */
public class ManageCrudEntitiesEvent {
}
```

El evento EditCrudEntityEvent indica que una entidad dada debe ser editada. Esto típicamente abrirá un formulario de ABM..

```
package ar.com.oxen.nibiru.crud.manager.api;

import ar.com.oxen.nibiru.conversation.api.Conversation;

public class EditCrudEntityEvent {
        private CrudEntity<?> entity;
        private Conversation conversation;

        public EditCrudEntityEvent(CrudEntity<?> entity, Conversation conversatio
                super();
                this.entity = entity;
                this.conversation = conversation;
        }

        public CrudEntity<?> getCrudEntity() {
                return entity;
```

```
        }

        public Conversation getConversation() {
                return conversation;
        }
}
```

Cuando finaliza la edición, se puede lanzar un ModifiedCrudEntityEvent para notificar que dicha instancia ha sido modificada. Por ejemplo, el presentador de lista de ABM escucha este evento para actualizar la lista.

```
package ar.com.oxen.nibiru.crud.manager.api;

public class ModifiedCrudEntityEvent {
        private CrudEntity<?> entity;

        public ModifiedCrudEntityEvent(CrudEntity<?> entity) {
                super();
                this.entity = entity;
        }

        public CrudEntity<?> getCrudEntity() {
                return entity;
        }
}
```

Finalmente, se puede disparar un ManageChildCrudEntitiesEvent para activar un ABM de entidades dependientes (en una relación padre-hijo).

```
package ar.com.oxen.nibiru.crud.manager.api;

/**
 * This is a generic event class for triggering entities management related to a
 * parent. The topic should be used in order to identify the entity to be
 * managed.
 */
public class ManageChildCrudEntitiesEvent {
        private String parentField;
        private Object parentEntity;

        public ManageChildCrudEntitiesEvent(String parentField, Object parentEnt
                super();
                this.parentField = parentField;
                this.parentEntity = parentEntity;
        }

        public String getParentField() {
```

```
                return parentField;
        }

        public Object getParentEntity() {
                return parentEntity;
        }
}
```

15.2. Servicios de interfaz de usuario

En el proyecto ar.com.oxen.nibiru.crud.ui.api se encuentran las interfaces para vistas y presentadores de las pantallas de ABM.

Dichas interfaces deben ser instanciadas por implementaciones del factory de presentadores:

```
package ar.com.oxen.nibiru.crud.ui.api;

import ar.com.oxen.nibiru.crud.manager.api.CrudManager;
import ar.com.oxen.nibiru.crud.manager.api.EditCrudEntityEvent;
import ar.com.oxen.nibiru.crud.ui.api.form.CrudFormView;
import ar.com.oxen.nibiru.crud.ui.api.list.CrudListView;
import ar.com.oxen.nibiru.ui.api.mvp.Presenter;

/**
 * CRUD presenter factory.
 */
public interface CrudPresenterFactory {
        /**
         * Builds a presenter for CRUD list.
         *
         * @param crudManager
         *              The CRUD manager
         * @return The presenter
         */
        Presenter<CrudListView> buildListPresenter(CrudManager<?> crudManager);

        /**
         * Builds a presenter for CRUD list which is filtered by a parent value.
         *
         * @param crudManager
         *              The CRUD manager
         * @param parentField
         *              The field used in order to filter the parent value.
         * @param parentValue
         *              The parent value.
```

```
         * @return The presenter
         */
        Presenter<CrudListView> buildListPresenter(CrudManager<?> crudManager,
                        String parentField, Object parentValue);

        /**
         * Builds a presenter for CRUD form.
         *
         * @param crudManager
         *              The CRUD manager
         * @return The presenter
         */
        Presenter<CrudFormView> buildFormPresenter(CrudManager<?> crudManager,
                        EditCrudEntityEvent event);
}
```

y del factory de vistas:

```
package ar.com.oxen.nibiru.crud.ui.api;

import ar.com.oxen.nibiru.crud.ui.api.form.CrudFormView;
import ar.com.oxen.nibiru.crud.ui.api.list.CrudListView;

/**
 * CRUD presenter factory.
 */
public interface CrudViewFactory {
        String I18N_FIELD_PREFIX = "ar.com.oxen.nibiru.crud.field.";
        String I18N_ACTION_PREFIX = "ar.com.oxen.nibiru.crud.action.";
        String I18N_ENTITY_PREFIX = "ar.com.oxen.nibiru.crud.entity.";
        String I18N_TAB_PREFIX = "ar.com.oxen.nibiru.crud.tab.";
        String I18N_ERROR_PREFIX = "ar.com.oxen.nibiru.crud.error.";

        /**
         * Builds the view for CRUD list.
         *
         * @return The view
         */
        CrudListView buildListView();

        /**
         * Builds the view for CRUD form.
         *
         * @return The view
         */
        CrudFormView buildFormView();
}
```

Existe una implementación genérica que se encuentra en el proyecto ar.com.oxen.nibiru.crud.ui.generic.

15.3. Utilidades

El bundle ar.com.oxen.nibiru.crud.utils contiene clases genéricas de utilidad para la creación de ABMs. Esto inlcuye:

- Implementaciones simples de CrudField y CrudAction.
- Action extensions comunes.
- Una clase base para configurar módulos de ABM (AbstractCrudModuleConfigurator).

La clase AbstractCrudModuleConfigurator provee los siguientes métodos:

- addCrud: Agrega un ABM independiente, que es activado desde el menú de la aplicación. El método registra los puntos de extensión para el menú y las acciones. Además registra en el bus de eventos los listeners necesarios para navegación.
- addChildCrud: Agrega un ABM hijo, que es disparado desde el menú contextual del ABM padre. De manera similar, regista las extensiones y los listeners necesarios.

```
package ar.com.oxen.nibiru.crud.utils;

import java.util.LinkedList;
import java.util.List;

import ar.com.oxen.commons.eventbus.api.EventHandler;
import ar.com.oxen.nibiru.crud.manager.api.CrudActionExtension;
import ar.com.oxen.nibiru.crud.manager.api.CrudManager;
import ar.com.oxen.nibiru.crud.manager.api.EditCrudEntityEvent;
import ar.com.oxen.nibiru.crud.manager.api.ManageChildCrudEntitiesEvent;
import ar.com.oxen.nibiru.crud.manager.api.ManageCrudEntitiesEvent;
import ar.com.oxen.nibiru.crud.ui.api.CrudPresenterFactory;
import ar.com.oxen.nibiru.crud.ui.api.CrudViewFactory;
import ar.com.oxen.nibiru.module.utils.AbstractModuleConfigurator;
import ar.com.oxen.nibiru.ui.api.extension.MenuItemExtension;
import ar.com.oxen.nibiru.ui.utils.extension.SimpleMenuItemExtension;
import ar.com.oxen.nibiru.ui.utils.mvp.SimpleEventBusClickHandler;

public abstract class AbstractCrudModuleConfigurator extends
                AbstractModuleConfigurator<CrudViewFactory, CrudPresenterFactory
```

```
private List<EventHandler<?>> registeredHandlers = new LinkedList<EventH

int menuPos = 0;

/**
 * Adds a CRUD menu
 */
protected <K> void addCrudMenu(String menuName, String parentMenuExtensi
                CrudManager<K> crudManager) {
        this.registerMenu(menuName, parentMenuExtension, crudManager);
}

/**
 * Adds a CRUD without a menu option
 */
protected <K> void addCrud(CrudManager<K> crudManager,
                CrudActionExtension<K> crudActionExtension) {
        this.registerManageEntityEvent(crudManager);
        this.registerActions(crudManager, crudActionExtension);
        this.registerEditEntityEvent(crudManager);
}

/**
 * Adds a CRUD with a menu option
 */
protected <K> void addCrudWithMenu(String menuName,
                String parentMenuExtension, CrudManager<K> crudManager,
                CrudActionExtension<K> crudActionExtension) {
        this.addCrudMenu(menuName, parentMenuExtension, crudManager);
        this.addCrud(crudManager, crudActionExtension);
}

/**
 * Adds a child menu CRUD menu option
 */
protected <T> void addChildCrudMenu(String menuName,
                CrudManager<?> parentCrudManager, String parentField,
                CrudManager<T> childCrudManager) {

        this.registerManageChildrenAction(menuName, parentCrudManager,
                        childCrudManager, parentField);
}

/**
 * Adds a child menu CRUD without a menu option
 */
```

```
protected <T> void addChildCrud(CrudManager<?> parentCrudManager,
                CrudManager<T> childCrudManager,
                CrudActionExtension<T> childCrudActionExtension) {

        this.registerActions(childCrudManager, childCrudActionExtension)
        this.registerManageChildEntitiesEvent(parentCrudManager,
                        childCrudManager);
        this.registerEditEntityEvent(childCrudManager);
}

/**
 * Adds a child menu CRUD with a menu option
 */
protected <T> void addChildCrudWithMenu(String menuName,
                CrudManager<?> parentCrudManager, String parentField,
                CrudManager<T> childCrudManager,
                CrudActionExtension<T> childCrudActionExtension) {

        this.addChildCrudMenu(menuName, parentCrudManager, parentField,
                        childCrudManager);
        this.addChildCrud(parentCrudManager, childCrudManager,
                        childCrudActionExtension);
}

@Override
public void shutdown() {
        super.shutdown();
        for (EventHandler<?> handler : this.registeredHandlers) {
                this.getEventBus().removeHandler(handler);
        }
}

private void registerMenu(String menuName, String parentMenuExtension,
                CrudManager<?> crudManager) {
        this.registerExtension(new SimpleMenuItemExtension(menuName, men
                        new SimpleEventBusClickHandler(this.getEventBus(
                                        ManageCrudEntitiesEvent.class, c
                                                        .getEntityTypeNa
                        MenuItemExtension.class);
}

private <K> void registerActions(CrudManager<K> crudManager,
                CrudActionExtension<K> crudActionExtension) {
        this.registerExtension(crudActionExtension, crudManager
                        .getEntityTypeName(), CrudActionExtension.class)
}
```

```
private <K> void registerManageChildrenAction(String menuName,
                CrudManager<?> parentCrudManager,
                final CrudManager<?> childCrudManager, String parentFiel
        this.registerExtension(new ManageChildrenCrudActionExtension<Obj
                        menuName, parentField, childCrudManager.getEntity
                        this.getEventBus()), parentCrudManager.getEntityT
                        CrudActionExtension.class);
}

private void registerManageEntityEvent(final CrudManager<?> crudManager)
        this.addEventHandler(ManageCrudEntitiesEvent.class,
                        new EventHandler<ManageCrudEntitiesEvent>() {

                                @Override
                                public void onEvent(ManageCrudEntitiesEv
                                        activate(getViewFactory().buildL
                                                        getPresenterFact

                                }
                        }, crudManager.getEntityTypeName());
}

private void registerEditEntityEvent(final CrudManager<?> crudManager) {
        this.addEventHandler(EditCrudEntityEvent.class,
                        new EventHandler<EditCrudEntityEvent>() {

                                @Override
                                public void onEvent(EditCrudEntityEvent
                                        activate(getViewFactory().buildFo
                                                        getPresenterFact

                                }
                        }, crudManager.getEntityTypeName());
}

private void registerManageChildEntitiesEvent(
                CrudManager<?> parentCrudManager,
                final CrudManager<?> childCrudManager) {
        this.addEventHandler(ManageChildCrudEntitiesEvent.class,
                        new EventHandler<ManageChildCrudEntitiesEvent>()

                                @Override
                                public void onEvent(ManageChildCrudEntit
                                        activate(getViewFactory().buildL
                                                        getPresenterFact
```

```
                                        }
                                    }, childCrudManager.getEntityTypeName());
        }

        private <T> void addEventHandler(Class<T> eventClass,
                        EventHandler<T> handler, String topic) {
                this.registeredHandlers.add(handler);
                this.getEventBus().addHandler(eventClass, handler, topic);
        }
}
```

El proyecto ar.com.oxen.nibiru.crud.bean contiene clases de utilidad para implementaciones de ABM que utilicen beans, como por ejemplo una implementación de CrudEntity que delega en un bean (a través de BeanWrapper, de Oxen Java Commons). También tiene anotaciones para parametrizar el ABM directamente en el bean.

16. Reportes

TODO: Definir este módulo.

17. Workflow

TODO: Definir este módulo.

18. Bundles dinámicos

El objetivo del mecanismo de bundles dinámicos es proveer, en conjunto con el mecanismo de puntos de extensión, una forma de agregar o personalizar funcionalidad al sistema mientras éste esté corriendo.

OSGi ya provee un servicio para instalar bundles de manera dinámica. La idea del módulo provisto por Nibiru es brindar un mecanismo para que dichos bundles puedan ser almacenados en la base de datos y administrados desde la misma aplicación. Vale la pena notar que el servicio de Nibiru no es genérico, sino que está pensado concretamente para OSGi.

En conjunto con los distintos motores de scripting, es posible agregar funcionalidad sin recompilar ningún bundle.

18.1. Persistencia

El bundle ar.com.oxen.nibiru.dynamicbundle.domain contiene las clases de dominio utilizadas para persistir bundles. El bundle ar.com.oxen.nibiru.dynamicbundle.dao.api provee un DAO para leer y escribir dichas clases de dominio.

```
package ar.com.oxen.nibiru.dynamicbundle.dao.api;

import ar.com.oxen.commons.dao.api.ReadDao;
import ar.com.oxen.commons.dao.api.UpdateDao;
import ar.com.oxen.nibiru.dynamicbundle.domain.DynamicBundle;

/**
 * DAO for accessing dynamic bundles.
 */
public interface DynamicBundleDao extends ReadDao<DynamicBundle>,
                UpdateDao<DynamicBundle> {
}
```

Actualmente hay una implementación JPA de dicho DAO: ar.com.oxen.nibiru.dynamicbundle.dao.jpa.

TODO: definir mejor el modelo de datos de bundles dinámicos. Por ejemplo, sólo tiene dependencias por bundle, cuando en realidad es mejor poner dependencias por paquete. O también se podría pensar cómo hacer para que además de tener archivos de Spring pueda tener código Java compilado (binario).

TODO: La persistencia no se podría hacer por el manager del módulo de ABMs? Actualmente, para lo único que se está usando el DynamicBundleDao, desde SpringDynamicBundleManager, es para leer todos los bundles e inicializarlos al arrancar.

18.2. Negocio

El proyecto ar.com.oxen.nibiru.dynamicbundle.manager.api provee la interfaz DynamicBundleManager, que permite operar sobre bundles.

```
package ar.com.oxen.nibiru.dynamicbundle.manager.api;

import ar.com.oxen.nibiru.dynamicbundle.domain.DynamicBundle;

/**
 * Service for managing dynamic bundles.
 */
public interface DynamicBundleManager {
        /**
         * Deploys and stars a dynamic bundle.
         *
```

```
         * @param dynamicBundle
         *              The dynamic bundle
         */
        void start(DynamicBundle dynamicBundle);

        /**
         * Stops and undeploys a dynamic bundle.
         *
         * @param dynamicBundle
         *              The dynamic bundle
         */
        void stop(DynamicBundle dynamicBundle);
}
```

En ar.com.oxen.nibiru.dynamicbundle.manager.spring hay una implementación que se basa en Spring DM para acceder al BundleContext.

Para iniciar y detener servicios se utiliza el esquema de extensiones de acciones provisto por el módulo de ABM. La clase DynamicBundleStatusExtension del bundle ar.com.oxen.nibiru.dynamicbundle.module provee CrudActions para iniciar y detener un servicio dado (delegando en DynamicBundleManager).

```
package ar.com.oxen.nibiru.dynamicbundle.module;

import java.util.ArrayList;
import java.util.List;

import ar.com.oxen.nibiru.crud.utils.SimpleCrudAction;
import ar.com.oxen.nibiru.crud.manager.api.CrudAction;
import ar.com.oxen.nibiru.crud.manager.api.CrudActionExtension;
import ar.com.oxen.nibiru.crud.manager.api.CrudEntity;
import ar.com.oxen.nibiru.dynamicbundle.domain.DynamicBundle;
import ar.com.oxen.nibiru.dynamicbundle.manager.api.DynamicBundleManager;

/**
 * CRUD action extension for starting and stopping dynamic bundles. It delegates
 * on {@link DynamicBundleManager}.
 *
 */
public class DynamicBundleStatusExtension implements
                CrudActionExtension<DynamicBundle> {
        private List<CrudAction> actions;
        private final static String START = "start";
        private final static String STOP = "stop";
        private DynamicBundleManager dynamicBundleManager;

        public DynamicBundleStatusExtension() {
```

```
                super();
                this.actions = new ArrayList<CrudAction>(2);
                this.actions.add(new SimpleCrudAction(START, true, false, true,
                this.actions.add(new SimpleCrudAction(STOP, true, false, true, f
        }

        @Override
        public List<CrudAction> getActions() {
                return this.actions;
        }

        @Override
        public CrudEntity<DynamicBundle> performAction(CrudAction action,
                        CrudEntity<DynamicBundle> entity) {
                if (START.equals(action.getName())) {
                        this.dynamicBundleManager.start(entity.getEntity());
                        return null;
                } else if (STOP.equals(action.getName())) {
                        this.dynamicBundleManager.stop(entity.getEntity());
                        return null;
                } else {
                        throw new IllegalArgumentException("Invalid action: " +
                }
        }

        public void setDynamicBundleManager(
                        DynamicBundleManager dynamicBundleManager) {
                this.dynamicBundleManager = dynamicBundleManager;
        }
}
```

18.3. Interfaz de usuario

En el bundle ar.com.oxen.nibiru.dynamicbundle.module se configuran los eventos que activan los distintos presentadores y vistas. No hay un proyecto específico para UI ya que se basa en los servicios de UI provistos por el módulo de ABMs.

19. Log

TODO: Es necesario un servicio de log? o simplemente con commons loggin o SLF4J alcanza? OSGi tiene un servicio de log, se podría hacer un adaptador para SLF4J. Y seguir la misma lógic que con transacciones, JPA y DataSource.

Parte III

Licencia

El framework es distribuido bajo licencia Apache 2.0.