Referencia Nibiru 0.3

7 de noviembre de 2012

http://nibiru.googlecode.com

Parte I

Introducción

1. Objetivo del framework

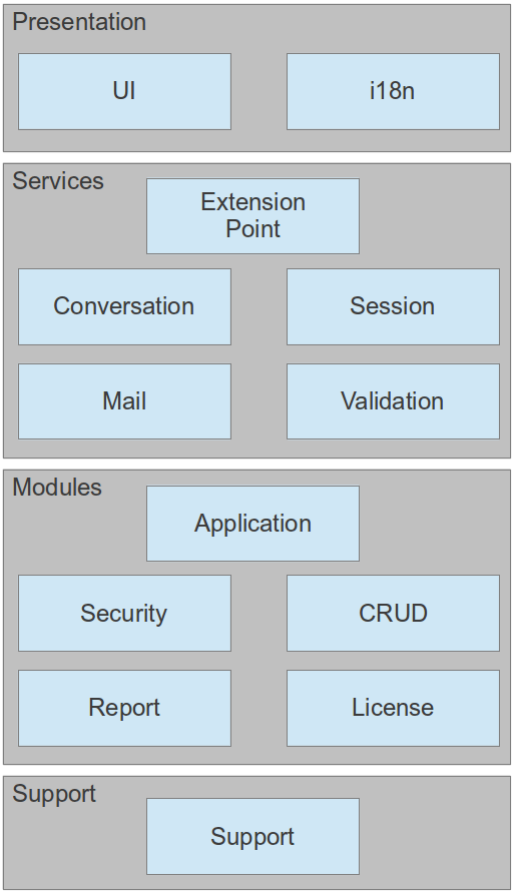
El objetivo es brindar un marco que facilite el desarrollo de aplicaciones modulares. Se establecen las siguientes metas para lograr dicho objetivo:

- 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 conceptos generales y decisiones de arquitectura tomadas.

2.1. Diagrama de alto nivel



2.2. Patrón IoC

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

2.3. 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.

2.4. 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. 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.

2.6. 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). De esta manera, Nibiru puede ser desplegado también en ambientes sin OSGi.

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. Esto creará un directorio ar.com.oxen.sample/ar.com.oxen.sample.targetplatform/target/platform con todos las dependencias del proyecto. Si no ocurre (o si se cambian las dependencias en algún momento), vaya a ese proyecto y ejecute "mvn compile" a fin de crear el target platform a partir de las dependencias Maven.
4. Importar los proyectos creados desde Eclipse. Se debe crear una variable de classpath M2_REPO apuntando al directorio m2/repository que se generó en su home directory.
5. En preferencias, activar el target platform "Nibiru Sample". Seleccionar la opción "reload" para que tome en cuenta los JARs descargados.
6. Ejecutar el launch de aplicación OSGi que se llama "Nibiru Sample". 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.

Puede ejecutar el ejemplo dentro de un entorno no-OSGi. El proyecto ar.com.oxen.nibiru.sample.springwebapp hace esto. Se ejecuta como un WAR convencional en un contenedor de servlets.
Puede descargar los binarios de aquí.

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/)*
4. *Crear un DSL con Builders*

Parte II

Estructura del proyecto

4. Subproyectos principales

La estrcutrua del proyecto Nibiru está organizada de una forma jerárquica. Dentro de esta estructura, los bundles principales son:

- ar.com.oxen.nibiru.application
- ar.com.oxen.nibiru.conversation
- ar.com.oxen.nibiru.crud
- ar.com.oxen.nibiru.extensionpoint
- ar.com.oxen.nibiru.i18n
- ar.com.oxen.nibiru.mail
- ar.com.oxen.nibiru.report
- ar.com.oxen.nibiru.security
- ar.com.oxen.nibiru.session
- ar.com.oxen.nibiru.support
- ar.com.oxen.nibiru.ui
- ar.com.oxen.nibiru.validation

Los mismos se encuentran en el directorio “main”.

5. Proyecto de ejemplo

El proyecto ar.com.oxen.nibiru.sample contiene una aplicación de ejemplo. El mismo se encuentra en el directorio "sample".

6. Categorización

6.1. 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".

6.2. División entre clases y servicios

El XML para exponer instancias de clases como servicios se encuentra en un bundle separado. De esta manera, se pueden exponer servicios de una manera diferente simplemente instalando otro bundle de servicio, con un XML personalizado (o incluso usando otra tecnología, como por ejemplo Peaberry). Y se pueden reutilizar las clases del bundle principal en la medida en la que esto sea necesario.

Los bundles que exportan servicio se llaman igual que el bundle que contiene la clase que lo implementa, pero con un sufijo ".service" en el nombre.

Parte III

Módulos

7. 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.main.MainView;
import ar.com.oxen.nibiru.ui.api.mvp.Presenter;

/**
 * Presenter factory for common application functionality.
 */
public interface ApplicationPresenterFactory {
        /**
         * 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.main.MainView;
```

```
/**
 * View factory for common application functionality.
 */
public interface ApplicationViewFactory {
        /**
         * Builds the view for main window.
         *
         * @return The view
         */
        MainView buildMainView();

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

7.1. Implementación genérica

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

Los bundles ar.com.oxen.nibiru.application.generic.presenter y ar.com.oxen.nibiru.application.generic.view proveen, respectivamente, las implementaciones genéricas para los presentadores y vistas de la aplicación.

8. 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 nombre. 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.

8.1. Implementación Spring DM

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.

8.2. Implementación genérica

El bundle ar.com.oxen.nibiru.extensionpoint.generic provee una implementación genérica del servicio de puntos de extensión que puede ser utilizada en ambientes que no soporten OSGi.

9. 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.

10. 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;
        }

        protected ExtensionPointManager getExtensionPointManager() {
                return extensionPointManager;
        }

        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();

        /**
         * @return Roles which this extension is available
         */
         String[] getAllowedRoles();
}
```

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();

        /**
         * @return Roles which this extension is available
         */
         String[] getAllowedRoles();

}
```

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.

El método getAllowedRoles indica los roles necesarios para ejecutar el menú. Estos roles se validan contra los servicios de seguridad. Si no se especifican los

roles (o si se devuelve null), no se lleva a cabo la validación (de modo que todo el mundo puede ejecutar el menú).

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

11. 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();
}
```

11.1. Implementación HTTP

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).

11.2. Integración con Spring

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>

...
```

12. 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 información 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.

12.1. Implementación genérica

El módulo ar.com.oxen.nibiru.conversation.generic provee una implementación genérica de los servicios de conversación.

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

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

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

public class GenericConversation implements Conversation {
```

```
private Set<ConversationTracker> trackers = new HashSet<ConversationTrac
private Map<String, Object> attributes = new HashMap<String, Object>();
private GenericConversationManager conversationManager;

public GenericConversation(GenericConversationManager conversationManage
        super();
        this.conversationManager = conversationManager;
}

@Override
public void end() {
        for (ConversationTracker tracker : this.trackers) {
                tracker.onEnd(this);
        }

        this.trackers.clear();
        this.attributes.clear();
}

@Override
public void cancel() {
        for (ConversationTracker tracker : this.trackers) {
                tracker.onCancel(this);
        }

        this.trackers.clear();
        this.attributes.clear();
}

@Override
public void registerTracker(ConversationTracker tracker) {
        this.trackers.add(tracker);
}

@Override
public <T> T execute(ConversationCallback<T> callback) {
        try {
                Conversation previousConversation = this.conversationMan
                                .getCurrentConversation();
                this.conversationManager.setCurrentConversation(this);
                T returnValue = callback.doInConversation(this);
                this.conversationManager
                                .setCurrentConversation(previousConversa
                return returnValue;
        } catch (Exception e) {
                throw new RuntimeException(e);
```

```
                }
        }

        @SuppressWarnings("unchecked")
        @Override
        public <T> T get(String key) {
                return (T) this.attributes.get(key);
        }

        @Override
        public void put(String key, Object value) {
                this.attributes.put(key, value);
        }

        @Override
        public void remove(String key) {
                this.attributes.remove(key);
        }
}

package ar.com.oxen.nibiru.conversation.generic;

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

public class GenericConversationManager implements ConversationFactory, Conversa
        private ThreadLocal<Conversation> currentConversation = new ThreadLocal<

        @Override
        public Conversation buildConversation() {
                return new GenericConversation(this);
        }

        @Override
        public Conversation getCurrentConversation() {
                return this.currentConversation.get();
        }

        void setCurrentConversation(Conversation conversation) {
                this.currentConversation.set(conversation);
        }

}
```

13. Persistencia

13.1. JPA

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.

13.2. Base de datos

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 y el bundle ar.com.oxen.nibiru.datasource.c3p0 provee una implementación con c3p0.

La configuración de conexión a la base de datos, así como la visibilidad del driver JDBC, se agregan también mediante fragmentos OSGi. Para más información, ver el proyecto ar.com.oxen.nibiru.sample.datasource.fragment. La implementación con c3p0 requiere de la creación de un fragmento extra, para que c3p0 tenga visibilidad sobre la clase del driver JDBC usado.

14. 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.
         */
        FormPanel buildFormPanel();

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

        /**
         * Builds an embedded.
         *
         * @return The embedded
         */
        Embedded buildEmbedded();
}
```

14.1. Vaadin implementation

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 mismo provee también una aplicación Vaadin específica para Nibiru:

```
package ar.com.oxen.nibiru.ui.vaadin.application;

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.commons.eventbus.api.EventHandler;
import ar.com.oxen.nibiru.application.api.ApplicationStartEvent;
import ar.com.oxen.nibiru.application.api.ApplicationThemeChangeEvent;
import ar.com.oxen.nibiru.i18n.api.LocaleHolder;

import com.vaadin.Application;

public class NibiruApplication extends Application {
        /**
         * Serial ID.
         */
        private static final long serialVersionUID = -8241304827319878154L;
        private EventBus eventBus;
        private LocaleHolder localeHolder;
        private EventHandler<ApplicationThemeChangeEvent> applicationThemeChange
```

```
        @Override
        public void init() {
                this.localeHolder.setLocale(this.getLocale());

                this.applicationThemeChangeEventHandler = new EventHandler<Appli
                        @Override
                        public void onEvent(ApplicationThemeChangeEvent event) {
                                setTheme(event.getTheme());
                        }
                };
                this.eventBus.addHandler(ApplicationThemeChangeEvent.class,
                                this.applicationThemeChangeEventHandler);

                this.eventBus.fireEvent(new ApplicationStartEvent());
        }

        @Override
        public void close() {
                this.eventBus.removeHandler(this.applicationThemeChangeEventHand
        }

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

        public void setLocaleHolder(LocaleHolder localeHolder) {
                this.localeHolder = localeHolder;
        }

}
```

Como se puede ver, si se desea cambiar el tema de Vaadin, se puede hacer disparando un evento ApplicationThemeChangeEvent en el bus.

Dado que la aplicación Vaadin no puede ser expuesta como servicio OSGi (dichos servicios son expuestos a través de interfaces Java y la aplicación Vaadin es una clase concreta), Nibiru provee una interfaz para tal componente:

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

import com.vaadin.Application;

/**
 * Interface for accessing Vaadin application from a service. Since
 * {@link Application} is not an interface, it can't be exposed as a service.
 */
```

```
public interface ApplicationAccessor {
        Application createApplication();

        Application getApplication();
}
```

Y, como es de esperar, una implementación simple:

```
package ar.com.oxen.nibiru.ui.vaadin.application;

import javax.servlet.http.HttpSession;

import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.nibiru.i18n.api.LocaleHolder;
import ar.com.oxen.nibiru.ui.vaadin.api.ApplicationAccessor;

import com.vaadin.Application;
import com.vaadin.terminal.gwt.server.WebApplicationContext;

public class SimpleApplicationAccessor implements ApplicationAccessor {
        private EventBus eventBus;
        private LocaleHolder localeHolder;

        @Override
        public Application createApplication() {
                NibiruApplication nibiruApplication = new NibiruApplication();
                nibiruApplication.setEventBus(this.eventBus);
                nibiruApplication.setLocaleHolder(this.localeHolder);
                return nibiruApplication;
        }

        @Override
        public Application getApplication() {
                WebApplicationContext context = WebApplicationContext
                                .getApplicationContext(this.currentSession());

                if (context.getApplications().size() > 0) {
                        return context.getApplications().iterator().next();
                } else {
                        throw new IllegalStateException("No Vaadin App on contex
                }
        }

        private ServletRequestAttributes currentAttributes() {
```

```
                return (ServletRequestAttributes) RequestContextHolder
                                .currentRequestAttributes();
        }

        private HttpSession currentSession() {
                return this.currentAttributes().getRequest().getSession(true);
        }

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

        public void setLocaleHolder(LocaleHolder localeHolder) {
                this.localeHolder = localeHolder;
        }
}
```

14.2. Utilidades de interfaz de usuario

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.

- ar.com.oxen.nibiru.ui.utils.dialog: Contiene clases para manejo de diálogos.
    - Por ejemplo, la clase DialogBuilder premite crear una ventana modal personalizada:

```
package ar.com.oxen.nibiru.ui.utils.dialog;

import ar.com.oxen.nibiru.ui.api.mvp.ClickHandler;
import ar.com.oxen.nibiru.ui.api.view.Button;
import ar.com.oxen.nibiru.ui.api.view.Label;
import ar.com.oxen.nibiru.ui.api.view.Panel;
import ar.com.oxen.nibiru.ui.api.view.ViewFactory;
import ar.com.oxen.nibiru.ui.api.view.Window;

public class DialogBuilder {
        private ViewFactory viewFactory;
        private Window window;
        private Panel messagePanel;;
        private Panel buttonPanel;;
```

```
public DialogBuilder(ViewFactory viewFactory) {
        super();
        this.viewFactory = viewFactory;

        this.window = viewFactory.buildWindow();
        this.window.setModal(true);

        this.messagePanel = viewFactory.buildVerticalPanel();
        this.window.addComponent(this.messagePanel);

        this.buttonPanel = viewFactory.buildHorizontalPanel();
        this.window.addComponent(this.buttonPanel);
}

public DialogBuilder title(String title) {
        this.window.setValue(title);
        return this;
}

public DialogBuilder message(String message) {
        Label<String> label = this.viewFactory.buildLabel(String.cla
        label.setValue(message);
        this.messagePanel.addComponent(label);
        return this;
}

public DialogBuilder button(String caption) {
        return this.button(caption, null);
}

public DialogBuilder button(String caption, final ClickHandler handl
        Button button = this.viewFactory.buildButton();
        button.setValue(caption);
        button.setClickHandler(new ClickHandler() {
                @Override
                public void onClick() {
                        window.close();
                        if (handler != null) {
                                handler.onClick();
                        }
                }
        });
        this.buttonPanel.addComponent(button);
        return this;
}
```

```
        public Window build() {
                return this.window;
        }
}
```

- ar.com.oxen.nibiru.ui.utils.extension: Provee implementaciones comunes de extensiones de interfaz de usuario.
  - SimpleMenuItemExtension es una implementación para ítems de menú:

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

import ar.com.oxen.nibiru.ui.api.extension.MenuItemExtension;
import ar.com.oxen.nibiru.ui.api.mvp.ClickHandler;

public class SimpleMenuItemExtension implements MenuItemExtension {
        private String name;
        private int position;
        private ClickHandler clickHandler;
        private String[] allowedRoles;

        public SimpleMenuItemExtension() {
                super();
        }

        public SimpleMenuItemExtension(String name, int position,
                        ClickHandler clickHandler) {
                this(name, position, clickHandler, null);
        }

        public SimpleMenuItemExtension(String name, int position,
                        ClickHandler clickHandler, String[] allowedRoles) {
                super();
                this.name = name;
                this.position = position;
                this.clickHandler = clickHandler;
                this.allowedRoles = allowedRoles;
        }

        @Override
        public String getName() {
                return this.name;
        }

        @Override
        public void onClick() {
```

```
            this.clickHandler.onClick();
        }

        public void setName(String name) {
            this.name = name;
        }

        public void setClickHandler(ClickHandler clickHandler) {
            this.clickHandler = clickHandler;
        }

        @Override
        public int getPosition() {
            return position;
        }

        public void setPosition(int position) {
            this.position = position;
        }

        @Override
        public String[] getAllowedRoles() {
            return this.allowedRoles;
        }
}
```

- SimpleSubMenuExtension, de manera similar, implementa una extensión para sub-menús:

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

import ar.com.oxen.nibiru.ui.api.extension.SubMenuExtension;

public class SimpleSubMenuExtension implements SubMenuExtension {
        private String name;
        private String extensionPoint;
        private int position;
        private String[] allowedRoles;

        public SimpleSubMenuExtension() {
            super();
        }
        public SimpleSubMenuExtension(String name, String extensionPoint,
                    int position) {
            this(name, extensionPoint, position, null);
        }
```

```
        public SimpleSubMenuExtension(String name, String extensionPoint,
                        int position, String[] allowedRoles) {
                super();
                this.name = name;
                this.extensionPoint = extensionPoint;
                this.position = position;
                this.allowedRoles = allowedRoles;
        }

        public void setName(String name) {
                this.name = name;
        }

        public void setExtensionPoint(String extensionPoint) {
                this.extensionPoint = extensionPoint;
        }

        @Override
        public String getName() {
                return this.name;
        }

        @Override
        public String getExtensionPoint() {
                return this.extensionPoint;
        }

        @Override
        public int getPosition() {
                return position;
        }

        public void setPosition(int position) {
                this.position = position;
        }
        @Override
        public String[] getAllowedRoles() {
                return this.allowedRoles;
        }
}
```

- ar.com.oxen.nibiru.ui.utils.mvp: Contiene clases de utilidad para implementar el patrón MVP.
  - AbstractEventBusClickHandler es una clase base para ClickHandlers que disparan eventos en el bus:

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

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.nibiru.ui.api.mvp.ClickHandler;

public abstract class AbstractEventBusClickHandler implements ClickHandler {
        private EventBus eventBus;

        public AbstractEventBusClickHandler() {
                super();
        }

        public AbstractEventBusClickHandler(EventBus eventBus) {
                super();
                this.eventBus = eventBus;
        }

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

        protected EventBus getEventBus() {
                return eventBus;
        }

}
```

- AbstractPresenter es una clase base para cualquier presentador:

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

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.nibiru.ui.api.mvp.ClickHandler;
import ar.com.oxen.nibiru.ui.api.mvp.HasCloseWidget;
import ar.com.oxen.nibiru.ui.api.mvp.Presenter;
import ar.com.oxen.nibiru.ui.api.mvp.View;

public abstract class AbstractPresenter<V extends View> implements Presenter
        private V view;
        private EventBus eventBus;

        protected AbstractPresenter(EventBus eventBus) {
                super();
                this.eventBus = eventBus;
        }
```

```
        @Override
        public void setView(V view) {
                this.view = view;
        }

        protected V getView() {
                return view;
        }

        protected EventBus getEventBus() {
                return eventBus;
        }

        protected void configureClose(HasCloseWidget hasCloseWidget) {
                hasCloseWidget.getCloseHandler().setClickHandler(new ClickHa
                        @Override
                        public void onClick() {
                                getView().close();
                        }
                });
        }
}
```

- HasValueI18nDecorator encapsula una instancia de HasValue<String> y realiza la traducción del texto mediante los servicios de internacionalización:

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

import ar.com.oxen.nibiru.i18n.api.MessageSource;
import ar.com.oxen.nibiru.ui.api.mvp.HasValue;

public class HasValueI18nDecorator implements HasValue<String> {
        private HasValue<String> decorated;
        private MessageSource messageSource;
        private String code;

        public HasValueI18nDecorator(HasValue<String> decorated,
                        MessageSource messageSource) {
                super();
                this.decorated = decorated;
                this.messageSource = messageSource;
        }

        @Override
        public String getValue() {
```

```
                return this.code;
        }

        @Override
        public void setValue(String value) {
                this.code = value;
                this.decorated.setValue(this.messageSource.getMessage(this.c
        }

}
```

- SimpleEventBusClickHandler es un manejador de evento de click que dispara un evento en el bus, con la clase y el tópico especificado:

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

import ar.com.oxen.commons.eventbus.api.EventBus;
import ar.com.oxen.commons.exception.api.ExceptionWrapper;

public class SimpleEventBusClickHandler extends AbstractEventBusClickHandler
        private Class<?> eventClass;
        private String topic;

        public SimpleEventBusClickHandler() {
                super();
        }

        public SimpleEventBusClickHandler(EventBus eventBus, Class<?> eventC
                        String topic) {
                super(eventBus);
                this.eventClass = eventClass;
                this.topic = topic;
        }

        @Override
        public void onClick() {
                try {
                        this.getEventBus().fireEvent(this.eventClass.newInst
                                        this.topic);
                } catch (InstantiationException e) {
                        throw new ExceptionWrapper(e);
                } catch (IllegalAccessException e) {
                        throw new ExceptionWrapper(e);
                }
        }
```

```
        public void setEventClass(Class<?> eventClass) {
                this.eventClass = eventClass;
        }

        public void setTopic(String topic) {
                this.topic = topic;
        }
}
```

- ar.com.oxen.nibiru.ui.utils.view: Provee clases base para fedinir vistas.
  - AbstractAdapter representa un adaptador genérico de vistas

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

public class AbstractAdapter<T> {
        private T adapted;

        public AbstractAdapter(T adapted) {
                super();
                this.adapted = adapted;
        }

        public T getAdapted() {
                return adapted;
        }
}
```

  - AbstractWindowViewAdapter es una clase base para vistas basadas en Window:

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

public class AbstractAdapter<T> {
        private T adapted;

        public AbstractAdapter(T adapted) {
                super();
                this.adapted = adapted;
        }

        public T getAdapted() {
                return adapted;
        }
}
```

15. 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 username
         *              The user name
         * @param password
         *              The password
         * @throws BadCredentialsException
         *               If the user name and/or the password is not valid
         */
        void login(String username, String password) throws BadCredentialsExcept

        /**
         * 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);
}
```

15.1. Implementación basada en Spring Security

El proyecto ar.com.oxen.nibiru.security.spring provee implementaciones de los componentes de seguridad basándose en el framework Spring Security.

La clase SpringAuthenticationService realiza la autenticación delegando en la clase AuthenticationManager de Spring Security:

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

import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticatio
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;

import ar.com.oxen.nibiru.security.api.AuthenticationService;
import ar.com.oxen.nibiru.security.api.BadCredentialsException;
import ar.com.oxen.nibiru.security.manager.api.SecurityManager;
import ar.com.oxen.nibiru.security.manager.api.UserData;
import ar.com.oxen.nibiru.session.api.Session;

public class SpringAuthenticationService implements AuthenticationService {
        private Session session;
        final static String AUTHENTICATION_KEY = "ar.com.oxen.nibiru.security.sp
        private AuthenticationManager authenticationManager;
        private SecurityManager securityManager;
        private SessionProfile profile;

        @Override
        public void login(String username, String password)
                        throws BadCredentialsException {
                try {
                        Authentication authentication = this.authenticationManag
                                        .authenticate(new UsernamePasswordAuthe
                                                        username, password != nu
                        this.session.put(AUTHENTICATION_KEY, authentication);
```

```
                        UserData userData = this.securityManager.getUserData(user
                        this.profile.activate(userData.getUsername(),
                                        userData.getFirstName(), userData.getLast

                } catch (AuthenticationException e) {
                        this.profile.deactivate();
                        throw new BadCredentialsException();
                }
        }

        @Override
        public void logout() {
                this.session.remove(AUTHENTICATION_KEY);
        }

        @Override
        public String getLoggedUserName() {
                UsernamePasswordAuthenticationToken authentication = this.sessio
                                .get(AUTHENTICATION_KEY);
                return authentication.getName();
        }

        public void setSession(Session session) {
                this.session = session;
        }

        public void setAuthenticationManager(
                        AuthenticationManager authenticationManager) {
                this.authenticationManager = authenticationManager;
        }

        public void setSecurityManager(SecurityManager securityManager) {
                this.securityManager = securityManager;
        }

        public void setProfile(SessionProfile profile) {
                this.profile = profile;
        }
}
```

Por supuesto, se debería inyectar una instancia de AuthenticationManager en las instancias esta clase.

Ya que SpringAuthenticationService almacena la información de autenticación en la sesión de Nibiru, la clase SpringAuthorizationService simplemente lee las authorities desde tal sesión:

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

import static ar.com.oxen.nibiru.security.spring.SpringAuthenticationService.AUT

import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;

import ar.com.oxen.nibiru.security.api.AuthorizationService;
import ar.com.oxen.nibiru.session.api.Session;

public class SpringAuthorizationService implements AuthorizationService {
        private Session session;

        @Override
        public boolean isCallerInRole(String role) {
                Authentication authentication = this.session.get(AUTHENTICATION_

                if (authentication != null) {
                        for (GrantedAuthority authority : authentication.getAutl
                                if (role.equals(authority.getAuthority())) {
                                        return true;
                                }
                        }
                }
                return false;
        }

        public void setSession(Session session) {
                this.session = session;
        }
}
```

15.2. Módulo seguridad

El bundle ar.com.oxen.nibiru.security.db bundle provee una implementación de seguridad con:

- Un modelo de dominio basado en usuarios, roles y grupos.
- Administración sobre esas entidades utilizando el módulo de ABM.
- Una implementación de UserDetailsService de Spring Security. La misma es inyectada a un AuthenticationManager, y este último es expuesto a fin de ser inyectado en un SpringAuthenticationService del módulo ar.com.oxen.nibiru.security.spring.

El siguiente archivo de ocnfiguración muestra cómo se conectan los componentes:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:security="http://www.springframework.org/schema/security"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:lang="http:/
        xmlns:osgi="http://www.springframework.org/schema/osgi" xmlns:tx="http:/
        xsi:schemaLocation="http://www.springframework.org/schema/beans
␣␣␣␣␣␣␣␣http://www.springframework.org/schema/beans/spring-beans.xsd
␣␣␣␣␣␣␣␣http://www.springframework.org/schema/osgi␣http://www.springframework.or
␣␣␣␣http://www.springframework.org/schema/lang␣http://www.springframework.org/scl
␣␣␣␣http://www.springframework.org/schema/security␣http://www.springframework.or
␣␣␣␣http://www.springframework.org/schema/tx␣http://www.springframework.org/sche

        <import resource="classpath:/ar/com/oxen/nibiru/security/module/context.

        <osgi:service interface="ar.com.oxen.nibiru.i18n.api.MessageProvider"
                ref="securityMessageProvider" />

        <osgi:reference id="authenticationService"
                interface="ar.com.oxen.nibiru.security.api.AuthenticationService

        <osgi:reference id="securityViewFactory"
                interface="ar.com.oxen.nibiru.security.ui.api.SecurityViewFactor

        <osgi:reference id="securityPresenterFactory"
                interface="ar.com.oxen.nibiru.security.ui.api.SecurityPresenterF

        <osgi:reference id="entityManagerFactory"
                interface="javax.persistence.EntityManagerFactory" />

        <osgi:reference id="transactionManager"
                interface="org.springframework.transaction.PlatformTransactionMa

        <osgi:reference id="extensionPointManager"
                interface="ar.com.oxen.nibiru.extensionpoint.api.ExtensionPointM

        <osgi:reference id="eventBus"
                interface="ar.com.oxen.commons.eventbus.api.EventBus" />

        <osgi:reference id="crudViewFactory"
                interface="ar.com.oxen.nibiru.crud.ui.api.CrudViewFactory" />

        <osgi:reference id="crudPresenterFactory"
                interface="ar.com.oxen.nibiru.crud.ui.api.CrudPresenterFactory"

        <osgi:reference id="wrapperFactory"
                interface="ar.com.oxen.commons.bean.api.WrapperFactory" />
```

```
        <bean
                class="org.springframework.orm.jpa.support.PersistenceAnnotationB

</beans>
```

16. 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.

17. 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:

```java
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 acc
 * 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 f
         *         returns the code.
         */
        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 f
         *         the code is returned.
         */
        String getMessage(String code, Locale locale, Object... args);

        /**
         * 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 f
         *         returns null.
```

```
         */
        String findMessage(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 f
         *         null is returned.
         */
        String findMessage(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 f
         *         null is returned.
         */
        String getMessage(String code, Locale locale, Object... args);
}
```

17.1. Implementación genérica

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.

17.2. Integración con la sesión

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

18. Validación

El proyecto ar.com.oxen.nibiru.validation.api define el API de validación. incluye 2 interfaces principales.

- Validator, la cual representa un componente que puede realizar una validación:

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

/**
 * Interface for classes that can perform validation.
 *
 * @param <T>
 *            Validated object data type.
 */
public interface Validator<T> {
        /**
         * Validates an object.
         *
         * @param object
         *              The object to be valdiated.
         * @throws ValidationException
         *              If the validation is not met.
         */
        void validate(T object) throws ValidationException;
}
```

- Validatable, que representa un componente al cual se le pueden asociar validadores:

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

/**
 * Something that can be validated.
 *
 * @param <T>
 *            Validated object data type.
 */
public interface Validatable<T> {
        /**
         * Adss a validator.
         *
         * @param validator
         *            The validator
         */
        void addValidator(Validator<T> validator);

        /**
         * Removes a validator.
         *
         * @param validator
         *            The validator
         */
        void removeValidator(Validator<T> validator);

        /**
         * Validates the validatable data.
         *
         * @throws ValidationException
         *             If the validation is not met.
         */
        void validate() throws ValidationException;
}
```

18.1. Validadores genéricos

El proyecto ar.com.oxen.nibiru.validation.generic provee validadores que pueden reutilizarse en distintos proyectos.

- NotEmptyValidator, que verifica que el dato no sea nulo o "":

```
package ar.com.oxen.nibiru.validation.generic;

import ar.com.oxen.nibiru.validation.api.ValidationException;
import ar.com.oxen.nibiru.validation.api.Validator;
```

```
/**
 * Validatro for using on required fields.
 * The validated object:
 *  - Must not be null
 *  - It String representation must not be "" neither spaces.
 */
public class NotEmptyValidator implements Validator<Object> {
        private String errorCode = "required";

        public NotEmptyValidator() {
                super();
        }

        public NotEmptyValidator(String errorCode) {
                super();
                this.errorCode = errorCode;
        }

        @Override
        public void validate(Object object) throws ValidationException {
                if (object == null || object.toString().trim().equals("")) {
                        throw new ValidationException(errorCode);
                }
        }
}
```

- RegexpValidator, el cual verifica el dato contra una expresión regular:

```
package ar.com.oxen.nibiru.validation.generic;

import java.util.regex.Pattern;

import ar.com.oxen.nibiru.validation.api.ValidationException;
import ar.com.oxen.nibiru.validation.api.Validator;

/**
 * Regexp-based validator. Validated must be an String
 */
public class RegexpValidator implements Validator<String> {
        private Pattern pattern;
        private String errorCode;

        public RegexpValidator(String regexp, String errorCode) {
                super();
                this.pattern = Pattern.compile(regexp);
```

```
                this.errorCode = errorCode;
        }

        @Override
        public void validate(String object) throws ValidationException {
                if (!pattern.matcher(object).matches()) {
                        throw new ValidationException(errorCode);
                }
        }
}
```

19. 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.

19.1. 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();

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

    /**
     * Finds an entity by its ID.
     *
     * @return The entity
     */
    CrudEntity<T> findById(Object id);

    /**
     * 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).

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

import java.util.List;

/**
 * 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 the id value.
         *
         * @return The id
         */
        Object getId();

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


        /**
         * 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<?> 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<?> 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 {
               String GENERAL_TAB = "general";

               /**
                * 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();

    String[] getAllowedRoles();
}
```

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".

El método getAllowedRoles indica los roles necesarios para ejecutar la acción. Estos roles se validan contra los servicios de seguridad. Si no se especifican los roles (o si se devuelve null), no se lleva a cabo la validación (de modo que todo el mundo puede ejecutar la acción).

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);

        /**
         *
         * @return allowed roles
         */
        String[] getAllowedRoles();
}
```

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.

19.1.1. 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.

```java
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..

```java
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.

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

public class ModifiedCrudEntityEvent {
        private Object id;

        public ModifiedCrudEntityEvent(Object id) {
                super();
                this.id = id;
```

```
        }

        public Object getId() {
                return id;
        }
}
```

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;
        }
}
```

19.2. 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.

19.3. 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 menu with allowed roles
         */
        protected <K> void addCrudMenu(String menuName, String parentMenuExtensi
                        CrudManager<K> crudManager, String[] allowedRoles) {
```

```
        this.registerMenu(menuName, parentMenuExtension, crudManager, al
}

/**
 * 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.addCrudWithMenu(menuName, parentMenuExtension, crudManager,
}

/**
 * Adds a CRUD with a menu option and allowed roles
 */
protected <K> void addCrudWithMenu(String menuName,
                String parentMenuExtension, CrudManager<K> crudManager,
                CrudActionExtension<K> crudActionExtension, String[] allo
        this.addCrudMenu(menuName, parentMenuExtension, crudManager, allo
        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.addChildCrudMenu(menuName, parentCrudManager, parentField,
}

/**
 * Adds a child menu CRUD menu option and allowed roles
 */
protected <T> void addChildCrudMenu(String menuName,
```

```
                CrudManager<?> parentCrudManager, String parentField,
                CrudManager<T> childCrudManager, String[] allowedRoles)

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

/**
 * 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);
}

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

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

```
}

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

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

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

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

protected <K> void registerManageChildrenAction(String menuName,
                CrudManager<?> parentCrudManager,
                final CrudManager<?> childCrudManager, String parentFiel
        this.registerManageChildrenAction(menuName, parentCrudManager, c
}

protected <K> void registerManageChildrenAction(String menuName,
                CrudManager<?> parentCrudManager,
                final CrudManager<?> childCrudManager, String parentFiel
        this.registerExtension(new ManageChildrenCrudActionExtension<Obj
                        menuName, parentField, childCrudManager.getEntity
                        this.getEventBus(), allowedRoles), parentCrudMan
```

```
                        CrudActionExtension.class);
    }

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

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

                            }
                        }, crudManager.getEntityTypeName());
    }

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

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

                            }
                        }, crudManager.getEntityTypeName());
    }

    protected 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);
        }
}
```

19.4. ABMs basados en beans

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.

Por ejemplo, la siguiente clase muestra algunas anotaciones de beans:

```
package ar.com.oxen.nibiru.sample.domain;

import java.util.Set;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

import ar.com.oxen.nibiru.crud.bean.annotation.Action;
import ar.com.oxen.nibiru.crud.bean.annotation.Actions;
import ar.com.oxen.nibiru.crud.bean.annotation.Filter;
import ar.com.oxen.nibiru.crud.bean.annotation.Show;
import ar.com.oxen.nibiru.crud.bean.annotation.Widget;
import ar.com.oxen.nibiru.crud.manager.api.CrudAction;
import ar.com.oxen.nibiru.crud.manager.api.WidgetType;

@Entity
@Actions({
                @Action(name = CrudAction.NEW, requiresEntity = false, showInForm
                @Action(name = CrudAction.EDIT, requiresEntity = true, showInForm
                @Action(name = CrudAction.UPDATE, requiresEntity = true, showInF
                @Action(name = CrudAction.DELETE, requiresEntity = true, showInFo
@Filter("authz.isCallerInRole('admin') ? null : \"name like 'T%' \"")
public class Student {
        @Id
        @GeneratedValue
        @Show(order = 0)
```

```
@Widget(type = WidgetType.TEXT_FIELD, readonly = true)
private Integer id;

@Column
@Show(order = 10)
private String name;

@ManyToMany(mappedBy = "students")
@Show(order = 30, inList = false)
@Widget(type = WidgetType.MULTISELECT, tab = "courses")
private Set<Course> courses;

public Integer getId() {
        return id;
}

public void setId(Integer id) {
        this.id = id;
}

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public Set<Course> getCourses() {
        return courses;
}

public void setCourses(Set<Course> courses) {
        this.courses = courses;
}

@Override
public String toString() {
        return this.name;
}

@Override
public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
```

```
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Student other = (Student) obj;
        if (id == null) {
            if (other.id != null)
                return false;
        } else if (!id.equals(other.id))
            return false;
        return true;
    }
}
```

- @Actions/@Action: Define qué acciones pueden realizarse sobre la entidad o sobre la ventana de ABM.
- @Filter: Permite una expresión de filtrado arbitraria, la cual puede ser evaluada y pasada al CrudManager (por ejemplo, para ser usada en una consulta JPA). Esto resulta útil, por ejemplo, para definir seguridad a nivel de fila.
- @Show: Determina cómo y cuándo se muestra el campo.
- @Widget: Provee información acerca de cómo se generará el widget de interfaz de usuario donde se mostrará el campo.

19.5. Validación

La validación de campos de CRUD se puede realizar exponiendo un Validator como extensión.

El nombre del punto de extensión debe armarse concatenando el nombre de la entidad, un punto y el nombre del campo a validad. Por ejemplo:

```
...

this.registerExtension(new NotEmptyValidator(),
                       Subject.class.getName() +".description",
```

```
                Validator.class);
...
```

20. Reportes

El proyecto ar.com.oxen.nibiru.report.api define el API de reportes. El mismo incluye sólo una interfaz:

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

import java.io.InputStream;
import java.util.Map;

public interface Report {
        String EXTENSION_POINT_NAME = "ar.com.oxen.nibiru.reports";

        String getName();

        Iterable<String> getFormats();

        Iterable<ParameterDefinition> getParameterDefinitions();

        InputStream render(String format, Map<String, Object> parameters);

        interface ParameterDefinition {
                String getName();

                Class<?> getType();
        }
}
```

Dicha interfaz debe ser implementada por cualquier reporte, más allá del motor utilizado.

Dado que el reporte usualmente se expondrá como una extensión, se provee también un nombre para el punto de extensión correspondiente.

20.1. Implementación BIRT

El proyecto ar.com.oxen.nibiru.report.birt provee una implementación basada en BIRT:

```
package ar.com.oxen.nibiru.report.birt;
```

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Date;
import java.util.List;
import java.util.Map;

import org.eclipse.birt.report.engine.api.EngineConfig;
import org.eclipse.birt.report.engine.api.EngineConstants;
import org.eclipse.birt.report.engine.api.EngineException;
import org.eclipse.birt.report.engine.api.HTMLRenderOption;
import org.eclipse.birt.report.engine.api.IPDFRenderOption;
import org.eclipse.birt.report.engine.api.IParameterDefn;
import org.eclipse.birt.report.engine.api.IRenderOption;
import org.eclipse.birt.report.engine.api.IReportEngine;
import org.eclipse.birt.report.engine.api.IReportRunnable;
import org.eclipse.birt.report.engine.api.IRunAndRenderTask;
import org.eclipse.birt.report.engine.api.PDFRenderOption;
import org.eclipse.birt.report.engine.api.RenderOption;
import org.eclipse.birt.report.engine.api.ReportEngine;

import ar.com.oxen.nibiru.report.api.Report;

public class BirtReport implements Report {
        private IReportEngine engine;
        private IReportRunnable design;

        public BirtReport(String file) {
                super();
                try {
                        engine = new ReportEngine(new EngineConfig());
                        this.design = engine.openReportDesign(this
                                        .getReporFileInputStream(file));
                } catch (EngineException e) {
                        throw new BirtReportException(e);
                }
        }

        @Override
        public String getName() {
```

```
        return this.design.getDesignInstance().getDisplayName();
}

@Override
public Iterable<String> getFormats() {
        return Arrays.asList(new String[] { "pdf", "html" });
}

@Override
public Iterable<ParameterDefinition> getParameterDefinitions() {
        @SuppressWarnings("unchecked")
        Collection<IParameterDefn> parameterDefns = engine
                        .createGetParameterDefinitionTask(design).getPar
                                        false);
        List<ParameterDefinition> parameters = new ArrayList<Report.Para
                        parameterDefns.size());

        for (IParameterDefn parameterDefn : parameterDefns) {
                parameters.add(new IParameterDefnAdapter(parameterDefn))
        }

        return parameters;
}

@Override
public InputStream render(final String format,
                final Map<String, Object> parameters) {
        try {
                final PipedOutputStream output = new PipedOutputStream()
                InputStream input = new PipedInputStream(output);

                final ClassLoader classLoader = Thread.currentThread()
                                .getContextClassLoader();

                new Thread(new Runnable() {
                        @Override
                        public void run() {
                                render(format, parameters, output, classl
                        }
                }).start();

                return input;
        } catch (IOException e) {
                throw new BirtReportException(e);
        }
}
```

```
@SuppressWarnings("unchecked")
private void render(String format, Map<String, Object> parameters,
                OutputStream output, ClassLoader classLoader) {
        try {
                /* Create task to run and render the report */
                IRunAndRenderTask task = design.getReportEngine()
                                .createRunAndRenderTask(design);

                /* Set parent classloader for engine */
                task.getAppContext().put(
                                EngineConstants.APPCONTEXT_CLASSLOADER_K

                for (Map.Entry<String, Object> entry : parameters.entryS
                        task.setParameterValue(entry.getKey(), entry.get
                }

                final IRenderOption options = new RenderOption();
                options.setOutputFormat(format);

                options.setOutputStream(output);

                if (options.getOutputFormat().equalsIgnoreCase("html"))
                        final HTMLRenderOption htmlOptions = new HTMLRe
                                        options);
                        htmlOptions.setImageDirectory("img");
                        htmlOptions.setHtmlPagination(false);
                        htmlOptions.setHtmlRtLFlag(false);
                        htmlOptions.setEmbeddable(false);
                        htmlOptions.setSupportedImageFormats("PNG");
                } else if (options.getOutputFormat().equalsIgnoreCase("p
                        final PDFRenderOption pdfOptions = new PDFRender
                        pdfOptions.setOption(IPDFRenderOption.PAGE_OVERFI
                                        IPDFRenderOption.FIT_TO_PAGE_SIZ
                        pdfOptions.setOption(IPDFRenderOption.PAGE_OVERFI
                                        IPDFRenderOption.OUTPUT_TO_MULTI
                }

                task.setRenderOption(options);

                // run and render report
                task.run();

                task.close();

                output.flush();
```

```
                output.close();
        } catch (EngineException e) {
                throw new BirtReportException(e);
        } catch (IOException e) {
                throw new BirtReportException(e);
        }
}

private InputStream getReporFileInputStream(String file) {
        InputStream reportInputStream = this.getClass().getResourceAsStre
                        file);
        if (reportInputStream == null) {
                reportInputStream = Thread.currentThread().getContextCla
                                .getResourceAsStream(file);
        }
        if (reportInputStream == null) {
                throw new IllegalArgumentException("Invalid report file:
        }

        return reportInputStream;
}

private static class IParameterDefnAdapter implements ParameterDefinitio
        private IParameterDefn parameterDefn;

        public IParameterDefnAdapter(IParameterDefn parameterDefn) {
                super();
                this.parameterDefn = parameterDefn;
        }

        @Override
        public String getName() {
                return this.parameterDefn.getName();
        }

        @Override
        public Class<?> getType() {
                switch (this.parameterDefn.getDataType()) {
                case IParameterDefn.TYPE_BOOLEAN:
                        return Boolean.class;
                case IParameterDefn.TYPE_DATE:
                case IParameterDefn.TYPE_TIME:
                case IParameterDefn.TYPE_DATE_TIME:
                        return Date.class;
                case IParameterDefn.TYPE_DECIMAL:
                        return Double.class;
```

```
                        case IParameterDefn.TYPE_FLOAT:
                                return Float.class;
                        case IParameterDefn.TYPE_STRING:
                                return String.class;
                        default:
                                throw new IllegalStateException("Invalid parameter
                                                + this.parameterDefn.getDataType
                        }
                }
        }
}
```

La misma obtiene toda la información de reporte desde un archivo de reporte BIRT.

20.2. Integración con ABM

La integración con el módulo de ABMs puede hacerse con las clases del proyecto ar.com.oxen.nibiru.report.crud. El mismo provee:

- Un CrudManager para reportes:

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

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

import ar.com.oxen.nibiru.crud.manager.api.CrudEntity;
import ar.com.oxen.nibiru.crud.manager.api.CrudField;
import ar.com.oxen.nibiru.crud.manager.api.CrudManager;
import ar.com.oxen.nibiru.report.api.Report;
import ar.com.oxen.nibiru.extensionpoint.api.ExtensionPointManager;
import ar.com.oxen.nibiru.extensionpoint.api.ExtensionTracker;
import ar.com.oxen.nibiru.crud.utils.SimpleCrudField;

public class ReportCrudManager implements CrudManager<Report> {
        private List<Report> reports = new LinkedList<Report>();

        public ReportCrudManager(ExtensionPointManager extensionPointManager
                super();
                extensionPointManager.registerTracker(new ExtensionTracker<R
                        @Override
                        public void onRegister(Report extension) {
                                reports.add(extension);
```

```
                }

                @Override
                public void onUnregister(Report extension) {
                        reports.remove(extension);
                }
        }, Report.EXTENSION_POINT_NAME, Report.class);
}

@Override
public String getEntityTypeName() {
        return Report.class.getName();
}

@Override
public List<CrudField> getListFields() {
        List<CrudField> listFields = new ArrayList<CrudField>(1);
        listFields.add(new SimpleCrudField(ReportCrudEntity.REPORT_N
                        String.class, new SimpleCrudField.SimpleList
        return listFields;
}

@Override
public List<CrudEntity<Report>> findAll() {
        return this.toEntity(this.reports);
}

@Override
public CrudEntity<Report> findById(Object id) {
        return new ReportCrudEntity((Report) id);
}

@Override
public List<CrudEntity<Report>> findByfield(String field, Object val
        // TODO Filtrar la lista por los campos
        return this.toEntity(this.reports);
}

private List<CrudEntity<Report>> toEntity(List<Report> reports) {
        List<CrudEntity<Report>> entities = new ArrayList<CrudEntity
                        reports.size());
        for (Report report : reports) {
                entities.add(new ReportCrudEntity(report));
        }
        return entities;
}
```

```
}
```

- Un CrudEntity que encapsula un reporte:

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

import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import ar.com.oxen.nibiru.crud.manager.api.CrudEntity;
import ar.com.oxen.nibiru.crud.manager.api.CrudField;
import static ar.com.oxen.nibiru.crud.manager.api.CrudField.FormInfo.GENERAL
import ar.com.oxen.nibiru.crud.manager.api.WidgetType;
import ar.com.oxen.nibiru.crud.utils.SimpleCrudField;
import ar.com.oxen.nibiru.report.api.Report;
import ar.com.oxen.nibiru.report.api.Report.ParameterDefinition;

class ReportCrudEntity implements CrudEntity<Report> {
        final static String REPORT_NAME_FIELD = "reportName";
        final static String REPORT_FORMAT_FIELD = "reportFormat";
        private Report report;
        private String format;
        private Map<String, Object> parameters;

        public ReportCrudEntity(Report report) {
                super();
                this.report = report;
                this.parameters = new HashMap<String, Object>();
        }

        @Override
        public Object getId() {
                return report;
        }

        @Override
        public List<CrudField> getFormFields() {
                List<CrudField> formFields = new LinkedList<CrudField>();

                /* Format field */
                formFields.add(new SimpleCrudField(REPORT_FORMAT_FIELD, Stri
                                null, new SimpleCrudField.SimpleFormInfo(Wi
                                                false, 0, GENERAL_TAB)));
```

```
        /* Parameter fields */
        for (Report.ParameterDefinition paramDef : this.report
                        .getParameterDefinitions()) {
                formFields.add(new ParameterDefinitionAdapter(paramD
        }

        return formFields;
}

@Override
public Object getValue(CrudField field) {
        return this.getValue(field.getName());
}

@Override
public Object getValue(String fieldName) {
        if (REPORT_NAME_FIELD.equals(fieldName)) {
                return this.report.getName();
        } else if (REPORT_FORMAT_FIELD.equals(fieldName)) {
                return this.format;
        } else {
                return this.parameters.get(fieldName);
        }
}

@Override
public void setValue(CrudField field, Object value) {
        this.setValue(field.getName(), value);
}

@Override
public void setValue(String fieldName, Object value) {
        if (REPORT_NAME_FIELD.equals(fieldName)) {
                throw new IllegalArgumentException("Report name can'
                                + fieldName);
        } else if (REPORT_FORMAT_FIELD.equals(fieldName)) {
                this.format = (String) value;
        } else {
                this.parameters.put(fieldName, value);
        }
}

@Override
public Report getEntity() {
        return report;
}
```

```
@Override
public String getEntityTypeName() {
        return Report.class.getName();
}

@Override
public Iterable<?> getAvailableValues(CrudField field) {
        return this.getAvailableValues(field.getName());
}

@Override
public Iterable<?> getAvailableValues(String fieldName) {
        if (REPORT_FORMAT_FIELD.equals(fieldName)) {
                return this.report.getFormats();
        } else {
                throw new IllegalArgumentException(
                                "Field with no available values: " +
        }
}

private static class ParameterDefinitionAdapter implements CrudField
        private Report.ParameterDefinition paramDef;

        public ParameterDefinitionAdapter(ParameterDefinition paramD
                super();
                this.paramDef = paramDef;
        }

        @Override
        public String getName() {
                return this.paramDef.getName();
        }

        @Override
        public Class<?> getType() {
                return this.paramDef.getType();
        }

        @Override
        public ListInfo getListInfo() {
                return null;
        }

        @Override
        public FormInfo getFormInfo() {
```

```
                        return new SimpleCrudField.SimpleFormInfo(WidgetType
                                        false, 9999, GENERAL_TAB);
                }
        }
}
```

- Un CrudActionExtension que permite abrir y ejecutar un reporte:

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

import java.io.InputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import ar.com.oxen.commons.eventbus.api.EventBus;
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.crud.utils.SimpleCrudAction;
import ar.com.oxen.nibiru.report.api.Report;

public class ReportCrudActionExtension implements CrudActionExtension<Report
        private final static String OPEN_REPORT = "openReport";
        private final static String RUN_REPORT = "runReport";
        private EventBus eventBus;

        public ReportCrudActionExtension(EventBus eventBus) {
                super();
                this.eventBus = eventBus;
        }

        @Override
        public List<CrudAction> getActions() {
                List<CrudAction> actions = new ArrayList<CrudAction>(2);
                actions.add(new SimpleCrudAction(OPEN_REPORT, true, false, t
                                null));
                actions.add(new SimpleCrudAction(RUN_REPORT, true, true, fal
                                null));
                return actions;
        }

        @Override
        public CrudEntity<Report> performAction(CrudAction action,
                        CrudEntity<Report> entity) {
```

```
                if (action.getName().equals(OPEN_REPORT)) {
                        return entity;
                } else if (action.getName().equals(RUN_REPORT)) {
                        String format = (String) entity
                                        .getValue(ReportCrudEntity.REPORT_FO

                        Map<String, Object> parameters = new HashMap<String,

                        for (Report.ParameterDefinition parameterDef : entit
                                        .getParameterDefinitions()) {
                                parameters.put(parameterDef.getName(),
                                                entity.getValue(parameterDef
                        }

                        InputStream data = entity.getEntity().render(format,

                        this.eventBus.fireEvent(new ReportExecutedEvent(enti
                                        format, data));

                        return null;
                } else {
                        throw new IllegalArgumentException("Invlaid action:
                }
        }

        @Override
        public String[] getAllowedRoles() {
                return null;
        }
}
```

20.3. Módulo de reportes

El bundle ar.com.oxen.nibiru.report.module provee un módulo de reportes genérico, que usa el módulo de ABM para generar la interfaz de usuario responsable de ejecutar los reportes.

Además provee una vista genérica que muestra el reporte ejecutado.

21. Workflow

TODO: Definir este módulo.

22. Correo electrónico

El bundle ar.com.oxen.nibiru.mail.api provee el API para envío de correo electrónico. La clase MailMessage representa un correo electrónico:

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

import java.util.Collection;
import java.util.HashSet;

/**
 * A mail message.
 */
public class MailMessage {
        private String from;
        private Collection<String> to;
        private String subject;
        private String body;
        private String contentType;

        public MailMessage(String from, String subject, String body) {
                this(from, subject, body, "text/html");
        }

        public MailMessage(String from, String subject, String body,
                        String contentType) {
                super();
                this.from = from;
                this.subject = subject;
                this.body = body;
                this.contentType = contentType;
                this.to = new HashSet<String>();
        }

        public String getFrom() {
                return from;
        }

        public Collection<String> getTo() {
                return to;
        }

        public String getSubject() {
                return subject;
        }
```

```
        public String getBody() {
                return body;
        }

        public String getContentType() {
                return contentType;
        }

}
```

Tales mensajes pueden enviarse usando una instancia de MailService:

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

/**
 * A service for sending e-mails.
 */
public interface MailService {
        /**
         * Sends a mail message.
         *
         * @param message
         *              The mail message
         */
        void sendMail(MailMessage message);
}
```

22.1. Implementación JavaMail

El proyecto ar.com.oxen.nibiru.mail.javamail provee una implementación de MailService basada en JavaMail.

```
package ar.com.oxen.nibiru.mail.javamail;

import java.util.Properties;

import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
```

```
import ar.com.oxen.nibiru.mail.api.MailException;
import ar.com.oxen.nibiru.mail.api.MailMessage;
import ar.com.oxen.nibiru.mail.api.MailService;

public class JavaMailService implements MailService {
        private Session session;

        @Override
        public void sendMail(MailMessage message) {
                try {
                        MimeMessage mimeMessage = new MimeMessage(this.session);
                        mimeMessage.setFrom(new InternetAddress(message.getFrom(
                        for (String recipient : message.getTo()) {
                                mimeMessage.addRecipient(Message.RecipientType.T
                                                new InternetAddress(recipient));
                        }
                        mimeMessage.setSubject(message.getSubject());

                        if (message.getContentType() == null
                                        | message.getContentType().equals("text/
                                mimeMessage.setText(message.getBody());
                        } else {
                                mimeMessage.setContent(message.getBody(),
                                                message.getContentType());
                        }

                        Transport.send(mimeMessage);

                } catch (AddressException e) {
                        throw new MailException(e);
                } catch (MessagingException e) {
                        throw new MailException(e);
                }
        }

        public void setMailProperties(final Properties mailProperties) {
                this.session = Session.getDefaultInstance(mailProperties,
                                new Authenticator() {
                                        protected PasswordAuthentication getPass
                                                return new PasswordAuthenticatio
                                                                (String) mailPro
                                                                (String) mailPro

                                        }
                                });
        }
```

}

23. Licenciamiento

Este módulo permite gestionar la licencia del producto. Se basa en el módulo de licencia de Oxen Java Commons. Sin embargo, las clases de Oxen Java Commons sólo proveen componentes para requerimiento, autorización y validación de licencia. Para integrarse con Nibiru, se necesita funcionalidad extra, como por ejemplo, almacenamiento e interfaz de usuario. Tal funcionalidad es provista por este módulo y la misma es explicada en las siguientes secciones.

23.1. Almacenamiento de licencia

Una vez que se recibe una licencia, la misma debería ser almacenada en un medio persistente. Esto es lo que provee el módulo ar.com.oxen.nibiru.license.store.api.

La interfaz para leer y escribir licencias es LicenseStoreManager:

```
package ar.com.oxen.nibiru.license.store.api;

/**
 * Manage for loading and saving licenses.
 */
public interface LicenseStoreManager {
        String GENERIC_MODULE = "genericModule";

        /**
         * Loads a license for a given module.
         *
         * @param module
         *              The module
         * @return A String representing the license
         */
        String loadLicense(String module);

        /**
         * Saves a license for a given module.
         *
         * @param module
         *              The module
         * @param license
         *              A String representing the license
         */
        void saveLicense(String module, String license);
}
```

El proyecto ar.com.oxen.nibiru.license.store.jpa contiene una implementación basada en JPA de esta API.

23.2. Módulo de licencia

El proyecto ar.com.oxen.nibiru.license.module provee la interfaz gráfica para requerir licenciamiento. Se puede interactuar con ese módulo por medio del siguiente evento:

```
package ar.com.oxen.nibiru.license.module.event;

/**
 * Event for requesting a license.
 */
public class LicenseRequestEvent {
        private boolean showInvalidLicenseMessage;
        private Object callbackEvent;
        private String callbackTopic;

        /**
         * @param showInvalidLicenseMessage
         *              True if an invalid license message must be shown
         * @param callbackEvent
         *              The event to be fired once the license is loaded
         * @param callbackTopic
         *              The same, for topic
         */
        public LicenseRequestEvent(boolean showInvalidLicenseMessage,
                        Object callbackEvent, String callbackTopic) {
                super();
                this.showInvalidLicenseMessage = showInvalidLicenseMessage;
                this.callbackEvent = callbackEvent;
                this.callbackTopic = callbackTopic;
        }

        public boolean getShowInvalidLicenseMessage() {
                return showInvalidLicenseMessage;
        }

        public Object getCallbackEvent() {
                return callbackEvent;
        }

        public String getCallbackTopic() {
                return callbackTopic;
        }
```

}

Típicamente, se verificará la licencia utilizando las clases de Oxen Java Commons (después de leerla utilizando el API de almacenamiento de licencias). Si no se encuentra una licencia válida, se puede disparar este evento a fin de mostrar una ventana para solicitar una licencia válida. Después de que una licencia es ingresada, el evento con el tópico especificado es disparado. De este modo, la funcionalidad licenciada puede ejecutarse nuevamente.

Este módulo provee tanto vistas como presentadores para ar.com.oxen.commons.license.impl.DefaultLicenseInfo Por supuesto, si se necesita información de licencia personalizada, se deben crear vistas y presentadores para tal fin.

TODO: Crear un proyecto service, para que se puedan personalizar los servicios de vista y presenter

23.3. Interfaz de línea de comando

Para permitir que las licencias sean autorizadas, se provee una herramienta de línea de comando. La misma está incluida en el módulo ar.com.oxen.nibiru.license.cli.

Como en el caso anterior, esta herramienta se basa en información de licencia contenida en ar.com.oxen.commons.license.impl.DefaultLicenseInfo. Además, utiliza la dirección MAC para identificar el hardware. Si se requiere información de licencia y/o mecanismo de identificación diferentes, se debe crear un autorizador personalizado.

Parte IV

Despliegue

Una de las ventajas de desarrollar bajo el framework Nibiru es que se puede desplegar, de manera indistinta, en ambientes con o sin OSGi.

24. Despliegue ambientes OSGi

A fin de desplegar en ambientes OSGi, se debería partir la aplicación en bundles. Más allá de eso, hay algunos bundles específicos que podría ser necesario implementar. Los mismos son explicados en las siguientes secciones.

24.1. Proyecto aplicación Web

Se debe crear al menos un proyecto webapp, a fin de que dicha aplicación sea publicada. Por supuesto, si la aplicación está dividida en varios módulos, usualmente ellos compartirán la misma webapp.

Dentro de este proyecto, se puede personalizar la aplicación web. Por ejemplo, se puede cambiar el context path:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Nibiru sample webapp
Bundle-SymbolicName: ar.com.oxen.nibiru.sample.webapp
Bundle-Version: 0.3.0
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Web-ContextPath: /sample
Import-Package: ar.com.oxen.nibiru.ui.vaadin.api,
 com.vaadin,
 com.vaadin.service,
 org.springframework.osgi.web.context.support;version="2.0.0.M1",
 org.springframework.web.context;version="3.0.6.RELEASE",
 org.springframework.web.context.request;version="3.0.6.RELEASE"
Require-Bundle: com.vaadin;bundle-version="6.6.4"
```

La implementación actual usa Vaadin. Por este motivo, se deberían crear dos archivos dentro del directorio WEB-INF:

- web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.
         <display-name>Niburu test</display-name>

         <context-param>
                 <param-name>contextClass</param-name>
                 <param-value>org.springframework.osgi.web.context.support.Os
         </context-param>

         <listener>
                 <listener-class>org.springframework.web.context.ContextLoad
         </listener>

         <listener>
                 <listener-class>org.springframework.web.context.request.Requ
         </listener>
```

```
        <servlet>
                <servlet-name>Niburu</servlet-name>
                <servlet-class>ar.com.oxen.nibiru.ui.vaadin.servlet.NibiruA
                <init-param>
                        <param-name>widgetset</param-name>
                        <param-value>ar.com.oxen.nibiru.ui.vaadin.NibiruWid
                </init-param>
        </servlet>
        <servlet-mapping>
                <servlet-name>Niburu</servlet-name>
                <url-pattern>/*</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
                <servlet-name>Niburu</servlet-name>
                <url-pattern>/VAADIN/*</url-pattern>
        </servlet-mapping>
</web-app>
```

- applicationContext.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:lang="ht
        xmlns:osgi="http://www.springframework.org/schema/osgi" xmlns:aop="h
        xmlns:tx="http://www.springframework.org/schema/tx"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/osgi http://www.springframewor
        http://www.springframework.org/schema/lang http://www.springframework.or
        http://www.springframework.org/schema/aop http://www.springframework.org
        http://www.springframework.org/schema/tx http://www.springframework.org

        <osgi:reference id="applicationAccessor"
                interface="ar.com.oxen.nibiru.ui.vaadin.api.ApplicationAcces

</beans>
```

24.2. Fragmento Datasource

Este bundle provee la información necesaria para la conexión a la base de datos, agregando un archivo database.properties al bundle que provee el DataSource (por ejemplo, ar.com.oxen.nibiru.datasource.dbcp). Este fragmento debe agregar también dependencias a las clases del controlador de base de datos, de modo que el classloader del bundle que expone el DataSource pueda verlas.

24.3. Fragmento JPA

Este bundle agrega el archivo META-INF/persistence.xml al bundle JPA (por ejemplo, ar.com.oxen.nibiru.jpa.spring). También debe agregar dependencia con las clases de dominio, para que el class loader de JPA pueda verlas.

24.4. Fragmento ODA

Si su proyecto usa reportes BIRT basados en Open Data Access, se debe agregar este fragmento a fin de hacer que las clases del dirver de conexión a base de datos sean visibles para el bundle org.eclipse.birt.report.data.oda.jdbc.

25. Despliegue ambientes sin OSGi

El soporte para entornos sin OSGi se proporciona a través del proyecto ar.com.oxen.nibiru.standalone. Dicho proyecto contiene las dependencias necesarias para ejecutar el framework, de manera similar a cómo se hace en el proyecto ar.com.oxen.nibiru.targetplatform. Sin embargo, en este caso, su objetivo es que sea utilizado en un entorno Java estándar, tal como un contenedor de servlets.

El proyecto también contiene archivos de configuración de Spring para todos los módulos de Nibiru. El archivo ar/com/oxen/nibiru/standalone/context.xml consolida los archivos de configuraciónnecesarios para una aplicación típica.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:lang="http:/
       xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://w
       xmlns:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/lang http://www.springframework.org/sc
    http://www.springframework.org/schema/aop http://www.springframework.org/sch
    http://www.springframework.org/schema/security http://www.springframework.or
    http://www.springframework.org/schema/tx  http://www.springframework.org/sch

        <import resource="classpath:/ar/com/oxen/nibiru/standalone/vaadin.xml" />
        <import resource="classpath:/ar/com/oxen/nibiru/standalone/commons.xml"
        <import resource="classpath:/ar/com/oxen/nibiru/standalone/session.xml"
        <import resource="classpath:/ar/com/oxen/nibiru/standalone/conversation.
        <import resource="classpath:/ar/com/oxen/nibiru/standalone/i18n.xml" />
        <import resource="classpath:/ar/com/oxen/nibiru/standalone/application.x
        <import resource="classpath:/ar/com/oxen/nibiru/standalone/extension.xml
        <import resource="classpath:/ar/com/oxen/nibiru/standalone/crud.xml" />
```

```
    <import resource="classpath:/ar/com/oxen/nibiru/standalone/security.xml"
    <import resource="classpath:/ar/com/oxen/nibiru/standalone/database.xml"
    <import resource="classpath:/ar/com/oxen/nibiru/standalone/transaction.x
    <import resource="classpath:/ar/com/oxen/nibiru/standalone/jpa.xml" />
    <import resource="classpath:/ar/com/oxen/nibiru/standalone/scope.xml" />
    <!-- intentionally left out: mail service -->

</beans>
```

Sin embargo, no todos los módulos están incluidos en este archivo. Para explorar las configuraciones disponibles, vaya al paquete ar.com.oxen.nibiru.standalone.

Parte V

Licencia

El framework es distribuido bajo licencia Apache 2.0.